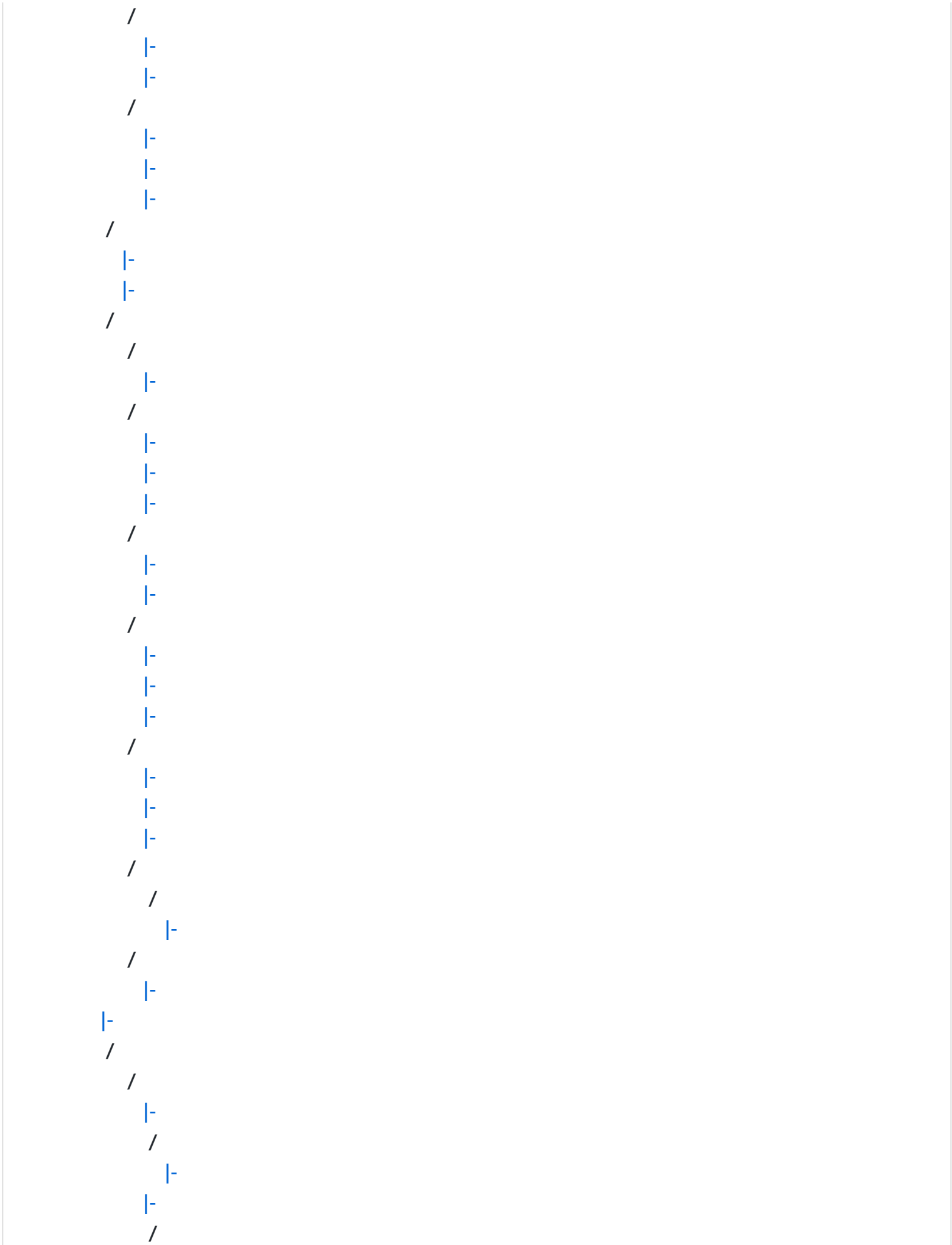
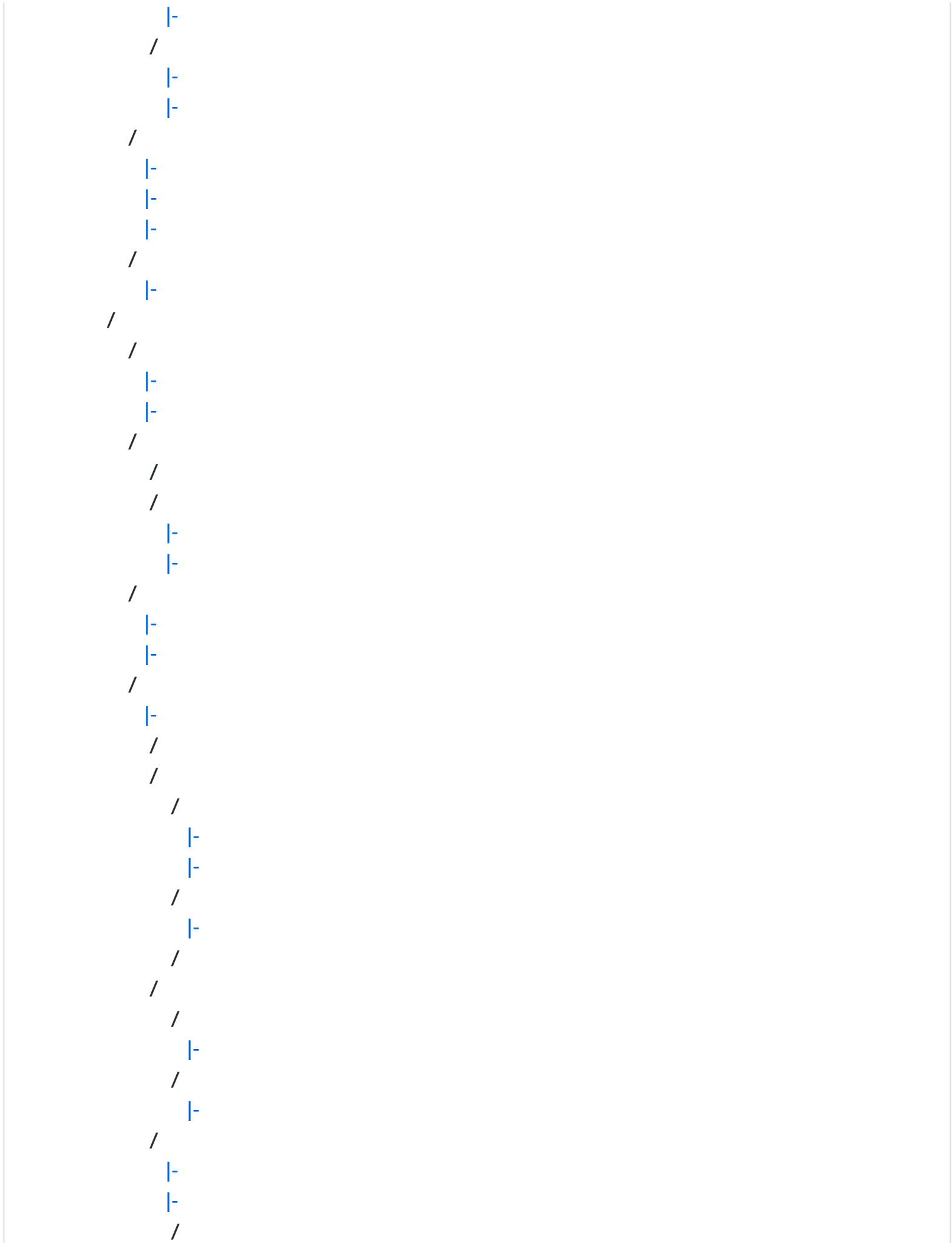
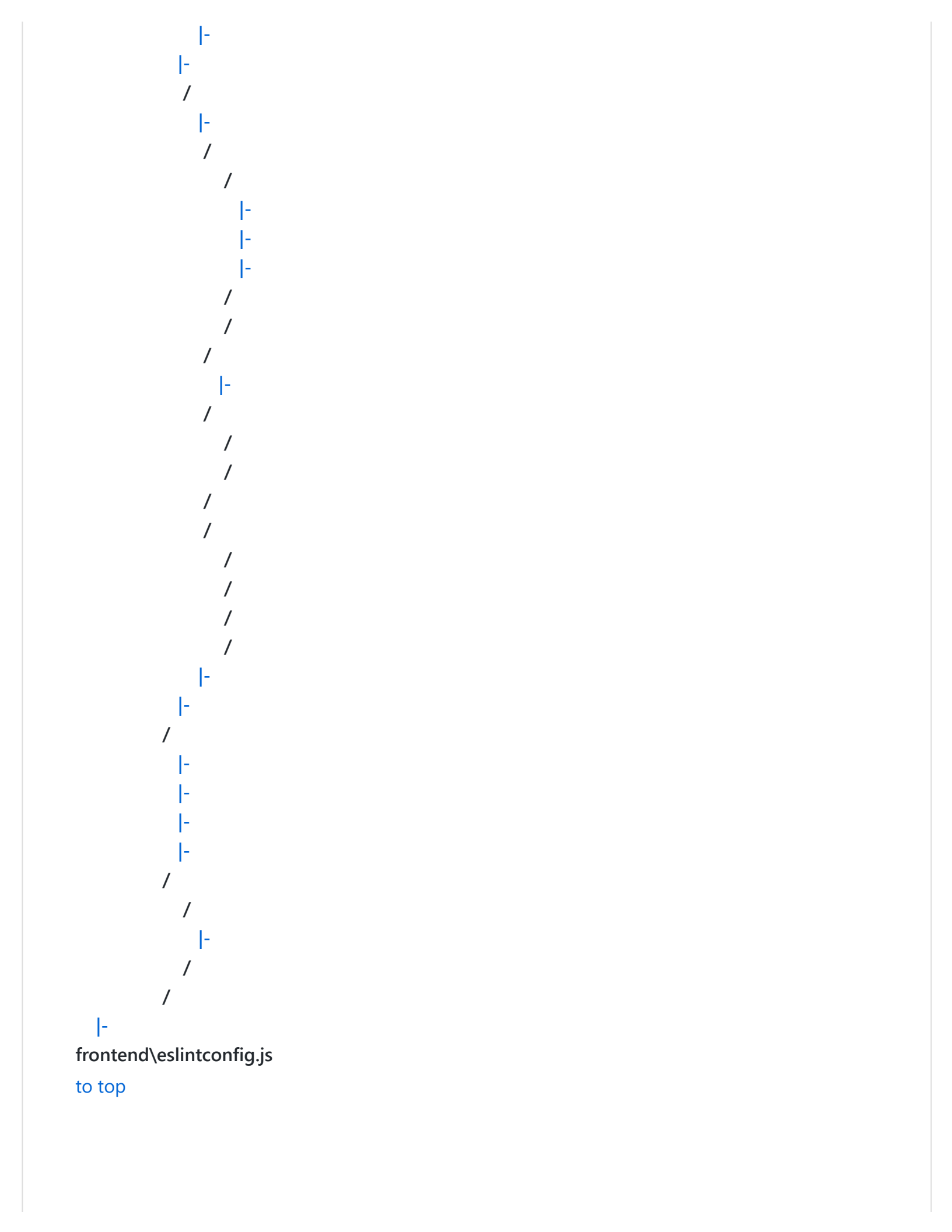
.

Contents

/
|-
|-
|-
/
|-
|-
/
|-
/
|-
/
/
|-
|-
/
|-
/
|-
|-
/
|-
|-
/
|-
/
/
|-
|-
|-
/
/
|-
/







to top

```
import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'
import tseslint from 'typescript-eslint'

export default tseslint.config(
  { ignores: ['dist'] },
  {
    extends: [js.configs.recommended, ...tseslint.configs.recommended],
    files: ['**/*.{ts,tsx}'],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
    },
    plugins: {
      'react-hooks': reactHooks,
      'react-refresh': reactRefresh,
    },
    rules: {
      ...reactHooks.configs.recommended.rules,
      'react-refresh/only-export-components': [
        'warn',
        { allowConstantExport: true },
      ],
    },
  },
)
```

frontend\indexhtml

to top

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React + TS</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

frontend\postcssconfig.js

to top

```
export default {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
}
```

frontend\public\vitesvg

to top

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
aria-hidden="true" role="img" class="iconify iconify--logos" width="31.88" height="32"
preserveAspectRatio="xMidYMid meet" viewBox="0 0 256 257"><defs><linearGradient
id="IconifyId1813088fe1fbc01fb466" x1="-.828%" x2="57.636%" y1="7.652%" y2="78.411%">
<stop offset="0%" stop-color="#41D1FF"></stop><stop offset="100%" stop-color="#BD34FE">
</stop></linearGradient><linearGradient id="IconifyId1813088fe1fbc01fb467" x1="43.376%"
x2="50.316%" y1="2.242%" y2="89.03%"><stop offset="0%" stop-color="#FFEA83"></stop>
<stop offset="8.333%" stop-color="#FFDD35"></stop><stop offset="100%" stop-
color="#FFA800"></stop></linearGradient></defs><path
fill="url(#IconifyId1813088fe1fbc01fb466)" d="M255.153 37.938L134.897 252.976c-2.483
4.44-8.862 4.466-11.382.048L.875 37.958c-2.746-4.814 1.371-10.646 6.827-9.67l120.385
21.517a6.537 6.537 0 0 0 2.322-.004l117.867-21.483c5.438-.991 9.574 4.796 6.877 9.62Z">
</path><path fill="url(#IconifyId1813088fe1fbc01fb467)" d="M185.432.063L96.44
17.501a3.268 3.268 0 0 0-2.634 3.014l-5.474 92.456a3.268 3.268 0 0 0 3.997
3.378l24.777-5.718c2.318-.535 4.413 1.507 3.936 3.838l-7.361 36.047c-.495 2.426 1.782
4.5 4.151 3.78l15.304-4.649c2.372-.72 4.652 1.36 4.15 3.788l-11.698 56.621c-.732 3.542
3.979 5.473 5.943 2.437l1.313-2.028l72.516-144.72c1.215-2.423-.88-5.186-3.54-4.672l-
25.505 4.922c-2.396.462-4.435-1.77-3.759-4.114l16.646-57.705c.677-2.35-1.37-4.583-
3.769-4.113Z"></path></svg>
```

frontend\READMEmd

to top

React + TypeScript + Vite

This template provides a minimal setup to get React working in Vite with HMR and some ESLint rules.

Currently, two official plugins are available:

- @vitejs/plugin-react (https://github.com/vitejs/vite-plugin-react/blob/main/packages/plugin-react/README.md) uses Babel (https://babeljs.io/) for Fast Refresh

- @vitejs/plugin-react-swc (https://github.com/vitejs/vite-plugin-react-swc) uses SWC (https://swc.rs/) for Fast Refresh

Expanding the ESLint configuration

If you are developing a production application, we recommend updating the configuration to enable type-aware lint rules:

```
export default tseslint.config({
  extends: [
    // Remove ...tseslint.configs.recommended and replace with this
    ...tseslint.configs.recommendedTypeChecked,
    // Alternatively, use this for stricter rules
    ...tseslint.configs.strictTypeChecked,
    // Optionally, add this for stylistic rules
    ...tseslint.configs.stylisticTypeChecked,
  ],
  languageOptions: {
    // other options...
    parserOptions: {
      project: ['./tsconfig.node.json', './tsconfig.app.json'],
      tsconfigRootDir: import.meta.dirname,
    },
  },
})
```

You can also install eslint-plugin-react-x (https://github.com/Rel1cx/eslint-react/tree/main/packages/plugins/eslint-plugin-react-x) and eslint-plugin-react-dom (https://github.com/Rel1cx/eslint-react/tree/main/packages/plugins/eslint-plugin-react-dom) for React-specific lint rules:

```
// eslint.config.js
import reactX from 'eslint-plugin-react-x'
import reactDom from 'eslint-plugin-react-dom'

export default tseslint.config({
  plugins: {
    // Add the react-x and react-dom plugins
    'react-x': reactX,
    'react-dom': reactDom,
  },
  rules: {
    // other rules...
    // Enable its recommended typescript rules
    ...reactX.configs['recommended-typescript'].rules,
    ...reactDom.configs.recommended.rules,
  },
})
```

frontend\src\Apptsx

to top

```
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom'
import { SocketProvider } from './context/SocketContext'
import Dashboard from './pages/Dashboard'
import Meeting from './pages/Meeting'
import './index.css'

function App() {
  return (
    <Router>
      <SocketProvider serverUrl="http://localhost:3000">
        <Routes>
          <Route path="/dashboard" element={<Dashboard />} />
          <Route path="/meeting/:roomId" element={<Meeting />} />
          <Route path="/" element={<Navigate to="/dashboard" replace />} />
        </Routes>
      </SocketProvider>
    </Router>
  )
}

export default App
```

frontend\src\assets\reactsvg

to top

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
aria-hidden="true" role="img" class="iconify iconify--logos" width="35.93" height="32"
preserveAspectRatio="xMidYMid meet" viewBox="0 0 256 228"><path fill="#00D8FF"
d="M210.483 73.824a171.49 171.49 0 0 0-8.24-2.597c.465-1.9.893-3.777 1.273-5.621c6.238-
30.281 2.16-54.676-11.769-62.708c-13.355-7.7-35.196.329-57.254 19.526a171.23 171.23 0 0
0-6.375 5.848a155.866 155.866 0 0 0-4.241-3.917C100.759 3.829 77.587-4.822 63.673
3.233C50.33 10.957 46.379 33.89 51.995 62.588a170.974 170.974 0 0 0 1.892 8.48c-
3.28.932-6.445 1.924-9.474 2.98C17.309 83.498 0 98.307 0 113.668c0 15.865 18.582 31.778
46.812 41.427a145.52 145.52 0 0 0 6.921 2.165a167.467 167.467 0 0 0-2.01 9.138c-5.354
28.2-1.173 50.591 12.134 58.266c13.744 7.926 36.812-.22 59.273-19.855a145.567 145.567 0
0 0 5.342-4.923a168.064 168.064 0 0 0 6.92 6.314c21.758 18.722 43.246 26.282 56.54
18.586c13.731-7.949 18.194-32.003 12.4-61.268a145.016 145.016 0 0 0-1.535-
6.842c1.62-.48 3.21-.974 4.76-1.488c29.348-9.723 48.443-25.443 48.443-41.52c0-15.417-
17.868-30.326-45.517-39.844Zm-6.365 70.984c-1.4.463-2.836.91-4.3 1.345c-3.24-10.257-
7.612-21.163-12.963-32.432c5.106-11 9.31-21.767 12.459-31.957c2.619.758 5.16 1.557 7.61
2.4c23.69 8.156 38.14 20.213 38.14 29.504c0 9.896-15.606 22.743-40.946 31.14Zm-10.514
20.834c2.562 12.94 2.927 24.64 1.23 33.787c-1.524 8.219-4.59 13.698-8.382 15.893c-8.067
4.67-25.32-1.4-43.927-17.412a156.726 156.726 0 0 1-6.437-5.87c7.214-7.889 14.423-17.06
21.459-27.246c12.376-1.098 24.068-2.894 34.671-5.345a134.17 134.17 0 0 1 1.386
6.193ZM87.276 214.515c-7.882 2.783-14.16 2.863-17.955.675c-8.075-4.657-11.432-22.636-
6.853-46.752a156.923 156.923 0 0 1 1.869-8.499c10.486 2.32 22.093 3.988 34.498
4.994c7.084 9.967 14.501 19.128 21.976 27.15a134.668 134.668 0 0 1-4.877 4.492c-9.933
8.682-19.886 14.842-28.658 17.94ZM50.35 144.747c-12.483-4.267-22.792-9.812-29.858-
15.863c-6.35-5.437-9.555-10.836-9.555-15.216c0-9.322 13.897-21.212 37.076-
29.293c2.813-.98 5.757-1.905 8.812-2.773c3.204 10.42 7.406 21.315 12.477 32.332c-5.137
11.18-9.399 22.249-12.634 32.792a134.718 134.718 0 0 1-6.318-1.979Zm12.378-84.26c-
4.811-24.587-1.616-43.134 6.425-47.789c8.564-4.958 27.502 2.111 47.463 19.835a144.318
144.318 0 0 1 3.841 3.545c-7.438 7.987-14.787 17.08-21.808 26.988c-12.04 1.116-23.565
2.908-34.161 5.309a160.342 160.342 0 0 1-1.76-7.887Zm110.427 27.268a347.8 347.8 0 0 0-
7.785-12.803c8.168 1.033 15.994 2.404 23.343 4.08c-2.206 7.072-4.956 14.465-8.193
22.045a381.151 381.151 0 0 0-7.365-13.322Zm-45.032-43.861c5.044 5.465 10.096 11.566
15.065 18.186a322.04 322.04 0 0 0-30.257-.006c4.974-6.559 10.069-12.652 15.192-
18.18ZM82.802 87.83a323.167 323.167 0 0 0-7.227 13.238c-3.184-7.553-5.909-14.98-8.134-
22.152c7.304-1.634 15.093-2.97 23.209-3.984a321.524 321.524 0 0 0-7.848 12.897Zm8.081
65.352c-8.385-.936-16.291-2.203-23.593-3.793c2.26-7.3 5.045-14.885 8.298-22.6a321.187
321.187 0 0 0 7.257 13.246c2.594 4.48 5.28 8.868 8.038 13.147Zm37.542 31.03c-5.184-
5.592-10.354-11.779-15.403-18.433c4.902.192 9.899.29 14.978.29c5.218 0 10.376-.117
15.453-.343c-4.985 6.774-10.018 12.97-15.028 18.486Zm52.198-57.817c3.422 7.8 6.306
15.345 8.596 22.52c-7.422 1.694-15.436 3.058-23.88 4.071a382.417 382.417 0 0 0 7.859-
13.026a347.403 347.403 0 0 0 7.425-13.565Zm-16.898 8.101a358.557 358.557 0 0 1-12.281
19.815a329.4 329.4 0 0 1-23.444.823c-7.967 0-15.716-.248-23.178-.732a310.202 310.202 0
0 1-12.513-19.846h.001a307.41 307.41 0 0 1-10.923-20.627a310.278 310.278 0 0 1 10.89-
20.637l-.001.001a307.318 307.318 0 0 1 12.413-19.761c7.613-.576 15.42-.876
23.31-.876H128c7.926 0 15.743.303 23.354.883a329.357 329.357 0 0 1 12.335
19.695a358.489 358.489 0 0 1 11.036 20.54a329.472 329.472 0 0 1-11 20.722Zm22.56-
122.124c8.572 4.944 11.906 24.881 6.52 51.026c-.344 1.668-.73 3.367-1.15 5.09c-10.622-
2.452-22.155-4.275-34.23-5.408c-7.034-10.017-14.323-19.124-21.64-27.008a160.789 160.789
0 0 1 5.888-5.4c18.9-16.447 36.564-22.941 44.612-18.3ZM128 90.808c12.625 0 22.86 10.235
22.86 22.86s-10.235 22.86-22.86 22.86s-22.86-10.235-22.86-22.86s10.235-22.86 22.86-
22.86Z"></path></svg>
```

frontend\src\components\video-call\JoinMeetingFormtsx

to top

```tsx
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';

interface JoinMeetingFormProps {
  onCreateMeeting?: (meetingName: string, userName: string) => Promise<string>;
}

const JoinMeetingForm: React.FC<JoinMeetingFormProps> = ({ onCreateMeeting }) => {
  const navigate = useNavigate();
  const [meetingId, setMeetingId] = useState('');
  const [userName, setUserName] = useState('');
  const [meetingName, setMeetingName] = useState('');
  const [isCreating, setIsCreating] = useState(false);
  const [error, setError] = useState('');
  const [isLoading, setIsLoading] = useState(false);

  const handleJoinMeeting = (e: React.FormEvent) => {
    e.preventDefault();

    if (!meetingId.trim()) {
      setError('Please enter a meeting ID');
      return;
    }

    if (!userName.trim()) {
      setError('Please enter your name');
      return;
    }

    // Store the username in localStorage for later use
    localStorage.setItem('userName', userName);

    // Navigate to the meeting room
    navigate(`/meeting/${meetingId}`);
  };

  const handleCreateMeeting = async (e: React.FormEvent) => {
    e.preventDefault();

    if (!meetingName.trim()) {
      setError('Please enter a meeting name');
      return;
    }

    if (!userName.trim()) {
      setError('Please enter your name');
      return;
    }

    setIsLoading(true);

    try {
```

```
      // If onCreateMeeting is provided, call it to create a new meeting
      if (onCreateMeeting) {
        const newMeetingId = await onCreateMeeting(meetingName, userName);

        // Store the username in localStorage for later use
        localStorage.setItem('userName', userName);

        // Navigate to the newly created meeting
        navigate(`/meeting/${newMeetingId}`);
      } else {
        // Generate a random meeting ID if no onCreateMeeting function is provided
        const randomId = Math.random().toString(36).substring(2, 10);

        // Store the username in localStorage for later use
        localStorage.setItem('userName', userName);

        // Navigate to the random meeting room
        navigate(`/meeting/${randomId}`);
      }
    } catch (err) {
      console.error('Error creating meeting:', err);
      setError('Failed to create meeting. Please try again.');
    } finally {
      setIsLoading(false);
    }
  };

  const toggleMode = () => {
    setIsCreating(!isCreating);
    setError('');
  };

  return (
    <div className="bg-white p-6 rounded-lg shadow-md w-full max-w-md">
      <h2 className="text-2xl font-bold mb-6 text-center">
        {isCreating ? 'Create a Meeting' : 'Join a Meeting'}
      </h2>

      {error && (
        <div className="mb-4 p-3 rounded bg-red-100 text-red-700 border border-red-
200">
          {error}
        </div>
      )}

      {isCreating ? (
        <form onSubmit={handleCreateMeeting} className="space-y-4">
          <div>
            <label htmlFor="meetingName" className="block text-sm font-medium text-
gray-700 mb-1">
              Meeting Name
            </label>
            <input
```

```
              type="text"
              id="meetingName"
              value={meetingName}
              onChange={(e) => setMeetingName(e.target.value)}
              className="w-full p-2 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500"
              placeholder="Enter meeting name"
              required
            />
          </div>

          <div>
            <label htmlFor="createUserName" className="block text-sm font-medium text-
gray-700 mb-1">
              Your Name
            </label>
            <input
              type="text"
              id="createUserName"
              value={userName}
              onChange={(e) => setUserName(e.target.value)}
              className="w-full p-2 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500"
              placeholder="Enter your name"
              required
            />
          </div>

          <button
            type="submit"
            disabled={isLoading}
            className="w-full py-2 px-4 bg-blue-600 text-white rounded hover:bg-blue-
700 focus:outline-none focus:ring-2 focus:ring-blue-500 focus:ring-offset-2
disabled:opacity-50"
          >
            {isLoading ? 'Creating...' : 'Create Meeting'}
          </button>
        </form>
      ) : (
        <form onSubmit={handleJoinMeeting} className="space-y-4">
          <div>
            <label htmlFor="meetingId" className="block text-sm font-medium text-gray-
700 mb-1">
              Meeting ID
            </label>
            <input
              type="text"
              id="meetingId"
              value={meetingId}
              onChange={(e) => setMeetingId(e.target.value)}
              className="w-full p-2 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500"
              placeholder="Enter meeting ID"
```

```
              required
            />
          </div>

          <div>
            <label htmlFor="joinUserName" className="block text-sm font-medium text-
gray-700 mb-1">
              Your Name
            </label>
            <input
              type="text"
              id="joinUserName"
              value={userName}
              onChange={(e) => setUserName(e.target.value)}
              className="w-full p-2 border border-gray-300 rounded focus:outline-none
focus:ring-2 focus:ring-blue-500"
              placeholder="Enter your name"
              required
            />
          </div>

          <button
            type="submit"
            className="w-full py-2 px-4 bg-blue-600 text-white rounded hover:bg-blue-
700 focus:outline-none focus:ring-2 focus:ring-blue-500 focus:ring-offset-2"
          >
            Join Meeting
          </button>
        </form>
      )}

      <div className="mt-4 text-center">
        <button
          type="button"
          onClick={toggleMode}
          className="text-blue-600 hover:text-blue-800 focus:outline-none"
        >
          {isCreating ? 'Join an existing meeting instead' : 'Create a new meeting
instead'}
        </button>
      </div>
    </div>
  );
};

export default JoinMeetingForm;
```

frontend\src\components\video-call\VideoCallRoomtsx

to top

```
import React, { useEffect, useRef, useState } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { Socket } from 'socket.io-client';
import useWebRTC from '../../hooks/useWebRTC';
import './VideoCallRoom.css';

interface VideoCallRoomProps {
  socket: Socket;
  userName?: string;
}

const VideoCallRoom: React.FC<VideoCallRoomProps> = ({ socket, userName }) => {
  const { roomId } = useParams<{ roomId: string }>();
  const navigate = useNavigate();
  const videoGridRef = useRef<HTMLDivElement>(null);
  const [message, setMessage] = useState<{ text: string; type: 'error' | 'success' |
'info' } | null>(null);
  const [isFullScreen, setIsFullScreen] = useState(false);
  const [participants, setParticipants] = useState<string[]>([]);
  const [remoteStreams, setRemoteStreams] = useState<Map<string, MediaStream>>(new
Map());

  const {
    initializeMedia,
    toggleVideo,
    toggleAudio,
    switchCamera,
    leaveMeeting,
    localStream,
    isVideoEnabled,
    isAudioEnabled,
    availableCameras,
    currentCameraId,
    remoteStreams: remoteStreamsFromHook,
    error
  } = useWebRTC(socket, {
    roomId,
    userName,
    onNewUser: (newUserId) => {
      setParticipants(prev => {
        // Check if participant already exists to prevent duplicates
        if (prev.includes(newUserId)) {
          return prev;
        }
        return [...prev, newUserId];
      });
      showMessage(`A new user joined the session`, 'info');
    },
    onUserDisconnected: (disconnectedUserId) => {
      setParticipants(prev => prev.filter(id => id !== disconnectedUserId));

      // Also remove any remote streams for this user
```

```
      setRemoteStreams(prev => {
        const updated = new Map(prev);
        updated.delete(disconnectedUserId);
        return updated;
      });

      showMessage(`A user left the session`, 'info');
    }
  });

  // Effect to handle remote streams from the useWebRTC hook
  useEffect(() => {
    if (remoteStreamsFromHook && remoteStreamsFromHook.size > 0) {
      console.log(`VideoCallRoom: Got ${remoteStreamsFromHook.size} remote streams`);
      setRemoteStreams(remoteStreamsFromHook);
    }
  }, [remoteStreamsFromHook]);

  // Effect to update the video grid when participants or streams change
  useEffect(() => {
    if (!videoGridRef.current) return;

    const gridElement = videoGridRef.current;

    // Clear previous videos
    while (gridElement.firstChild) {
      gridElement.removeChild(gridElement.firstChild);
    }

    // Add local video first
    if (localStream) {
      const videoContainer = document.createElement('div');
      videoContainer.className = 'video-container local-video';

      const video = document.createElement('video');
      video.srcObject = localStream;
      video.autoplay = true;
      video.playsInline = true;
      video.muted = true; // Mute local video to prevent feedback

      const nameLabel = document.createElement('div');
      nameLabel.className = 'name-label';
      nameLabel.textContent = `${userName || 'You'} (You)`;

      videoContainer.appendChild(video);
      videoContainer.appendChild(nameLabel);
      gridElement.appendChild(videoContainer);
    }

    // Add remote videos
    remoteStreams.forEach((stream, userId) => {
      if (stream) {
        console.log(`Adding video for user ${userId}`);
```

```
      const videoContainer = document.createElement('div');
      videoContainer.className = 'video-container remote-video';
      videoContainer.dataset.userId = userId;

      const video = document.createElement('video');
      video.srcObject = stream;
      video.autoplay = true;
      video.playsInline = true;

      const nameLabel = document.createElement('div');
      nameLabel.className = 'name-label';
      nameLabel.textContent = `Participant ${participants.indexOf(userId) + 1}`;

      videoContainer.appendChild(video);
      videoContainer.appendChild(nameLabel);
      gridElement.appendChild(videoContainer);

      // Ensure video plays
      video.play().catch(err => {
        console.error(`Error playing video for user ${userId}:`, err);
      });
    }
  });

  // Update grid layout based on number of videos
  const totalVideos = 1 + remoteStreams.size; // Local + remote videos
  gridElement.className = `video-grid videos-${totalVideos}`;

}, [localStream, remoteStreams, participants, userName]);

// Function to show messages to users
const showMessage = (text: string, type: 'error' | 'success' | 'info') => {
  setMessage({ text, type });

  // Automatically clear the message after 5 seconds
  setTimeout(() => {
    setMessage(null);
  }, 5000);
};

useEffect(() => {
  if (error) {
    showMessage(error, 'error');
  }
}, [error]);

useEffect(() => {
  // Initialize media when component mounts
  const setupCall = async () => {
    try {
      await initializeMedia();
    } catch (err) {
```

```
        showMessage('Failed to access camera and microphone', 'error');
        console.error(err);
      }
    };

    setupCall();

    // Clean up when component unmounts
    return () => {
      leaveMeeting();
    };
  }, [initializeMedia, leaveMeeting]);

  const handleToggleVideo = async () => {
    const isEnabled = await toggleVideo();
    showMessage(`Video is now ${isEnabled ? 'enabled' : 'disabled'}`, 'info');
  };

  const handleToggleAudio = () => {
    const isEnabled = toggleAudio();
    showMessage(`Audio is now ${isEnabled ? 'enabled' : 'disabled'}`, 'info');
  };

  const handleSwitchCamera = async (deviceId: string) => {
    try {
      await switchCamera(deviceId);
      showMessage('Camera switched successfully', 'success');
    } catch (err) {
      showMessage('Failed to switch camera', 'error');
    }
  };

  const handleLeaveCall = () => {
    leaveMeeting();
    navigate('/dashboard');
  };

  const toggleFullScreen = () => {
    if (!document.fullscreenElement) {
      videoGridRef.current?.requestFullscreen();
      setIsFullScreen(true);
    } else {
      document.exitFullscreen();
      setIsFullScreen(false);
    }
  };

  return (
    <div className="flex flex-col h-full">
      {/* Message display */}
      {message && (
        <div className={`fixed top-4 right-4 p-4 rounded shadow-lg z-50 ${
          message.type === 'error' ? 'bg-red-100 text-red-800 border-red-300' :
```

```
          message.type === 'success' ? 'bg-green-100 text-green-800 border-green-300' :
          'bg-blue-100 text-blue-800 border-blue-300'
        }`}>
          {message.text}
        </div>
      )}

      {/* Video grid */}
      <div className="flex-grow mb-4">
        <div
          ref={videoGridRef}
          className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4 h-full"
        >
          {/* Videos will be added here dynamically */}
        </div>
      </div>

      {/* Controls */}
      <div className="flex justify-center gap-4 p-4 bg-gray-100 rounded-lg">
        <button
          onClick={handleToggleVideo}
          className={`p-3 rounded-full ${isVideoEnabled ? 'bg-blue-500 text-white' :
'bg-red-500 text-white'}`}
          title={isVideoEnabled ? 'Turn off video' : 'Turn on video'}
        >
          <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6" fill="none"
viewBox="0 0 24 24" stroke="currentColor">
            {isVideoEnabled ? (
              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M15
10l4.553-2.276A1 1 0 0121 8.618v6.764a1 1 0 01-1.447.894L15 14M5 18h8a2 2 0 002-2V8a2 2
0 00-2-2H5a2 2 0 00-2 2v8a2 2 0 002 2z" />
            ) : (
              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M15
10l4.553-2.276A1 1 0 0121 8.618v6.764a1 1 0 01-1.447.894L15 14M5 18h8a2 2 0 002-2V8a2 2
0 00-2-2H5a2 2 0 00-2 2v8a2 2 0 002 2z M18.364 18.364A9 9 0 005.636 5.636" />
            )}
          </svg>
        </button>

        <button
          onClick={handleToggleAudio}
          className={`p-3 rounded-full ${isAudioEnabled ? 'bg-blue-500 text-white' :
'bg-red-500 text-white'}`}
          title={isAudioEnabled ? 'Mute audio' : 'Unmute audio'}
        >
          <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6" fill="none"
viewBox="0 0 24 24" stroke="currentColor">
            {isAudioEnabled ? (
              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M19
11a7 7 0 01-7 7m0 0a7 7 0 01-7-7m7 7v4m0 0H8m4 0h4m-4-8a3 3 0 01-3-3V5a3 3 0 116 0v6a3
3 0 01-3 3z" />
            ) : (
              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2}
```

```
d="M5.586 15H4a1 1 0 01-1-1v-4a1 1 0 011-1h1.586l4.707-4.707C10.923 3.663 12 4.109 12
5v14c0 .891-1.077 1.337-1.707.707L5.586 15z M17 14l2-2m0 0l2-2m-2 2l-2-2m2 2l2 2" />
            )}
          </svg>
        </button>

        {availableCameras.length > 1 && (
          <select
            onChange={(e) => handleSwitchCamera(e.target.value)}
            value={currentCameraId || ''}
            className="p-2 border rounded bg-white"
          >
            <option value="">Select Camera</option>
            {availableCameras.map((camera) => (
              <option key={camera.deviceId} value={camera.deviceId}>
                {camera.label || `Camera ${camera.deviceId.substring(0, 5)}`}
              </option>
            ))}
          </select>
        )}

        <button
          onClick={toggleFullScreen}
          className="p-3 rounded-full bg-gray-500 text-white"
          title={isFullScreen ? 'Exit full screen' : 'Enter full screen'}
        >
          <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6" fill="none"
viewBox="0 0 24 24" stroke="currentColor">
            {isFullScreen ? (
              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M9
9V4.5M9 9H4.5M15 9H19.5M15 9V4.5M15 15H19.5M15 15V19.5M9 15H4.5M9 15V19.5" />
            ) : (
              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M4
8V4m0 0h4M4 4l5 5m11-1V4m0 0h-4m4 0l-5 5M4 16v4m0 0h4m-4 0l5-5m11 5v-4m0 4h-4m4 0l-5 5"
/>
            )}
          </svg>
        </button>

        <button
          onClick={handleLeaveCall}
          className="p-3 rounded-full bg-red-600 text-white"
          title="Leave call"
        >
          <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6" fill="none"
viewBox="0 0 24 24" stroke="currentColor">
            <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M16
8l2-2m0 0l2-2m-2 2l-2-2m2 2l2 2M5 3a2 2 0 00-2 2v1c0 8.284 6.716 15 15 15h1a2 2 0 002-
2v-3.28a1 1 0 00-.684-.948l-4.493-1.498a1 1 0 00-1.21.502l-1.13 2.257a11.042 11.042 0
01-5.516-5.517l2.257-1.128a1 1 0 00.502-1.21L9.228 3.683A1 1 0 008.279 3H5z" />
          </svg>
        </button>
      </div>
```

```
      {/* Participants info */}
      <div className="mt-4 p-4 bg-gray-100 rounded-lg">
        <h3 className="text-lg font-medium mb-2">Participants ({participants.length +
1})</h3>
        <ul className="max-h-24 overflow-y-auto">
          <li className="py-1 px-2 bg-blue-100 rounded mb-1">
            You{userName ? ` (${userName})` : ''}
          </li>
          {participants.map((participantId) => (
            <li key={participantId} className="py-1 px-2 bg-gray-200 rounded mb-1">
              User {participantId.substring(0, 8)}
            </li>
          ))}
        </ul>
      </div>
    </div>
  );
};

export default VideoCallRoom;
```

frontend\src\context\SocketContexttsx

to top

```tsx
import React, { createContext, useContext, useEffect, useState } from 'react';
import { io, Socket } from 'socket.io-client';

interface SocketContextType {
  socket: Socket | null;
  isConnected: boolean;
  userId: string | null;
}

const SocketContext = createContext<SocketContextType>({
  socket: null,
  isConnected: false,
  userId: null
});

export const useSocket = () => useContext(SocketContext);

interface SocketProviderProps {
  children: React.ReactNode;
  serverUrl?: string;
}

export const SocketProvider: React.FC<SocketProviderProps> = ({
  children,
  serverUrl = 'http://localhost:3000' // Default to local server
}) => {
  const [socket, setSocket] = useState<Socket | null>(null);
  const [isConnected, setIsConnected] = useState(false);
  const [userId, setUserId] = useState<string | null>(null);

  useEffect(() => {
    // Initialize socket connection
    const socketInstance = io(serverUrl, {
      reconnectionAttempts: 5,
      reconnectionDelay: 1000,
      autoConnect: true,
      transports: ['websocket']
    });

    setSocket(socketInstance);

    // Setup event listeners
    const onConnect = () => {
      console.log('Connected to socket server');
      setIsConnected(true);
    };

    const onDisconnect = () => {
      console.log('Disconnected from socket server');
      setIsConnected(false);
    };
```

```
    const onConnectError = (error: Error) => {
      console.error('Connection error:', error);
      setIsConnected(false);
    };

    const onUserId = (id: string) => {
      console.log('Received user ID:', id);
      setUserId(id);
    };

    socketInstance.on('connect', onConnect);
    socketInstance.on('disconnect', onDisconnect);
    socketInstance.on('connect_error', onConnectError);
    socketInstance.on('userId', onUserId);

    // Cleanup on unmount
    return () => {
      socketInstance.off('connect', onConnect);
      socketInstance.off('disconnect', onDisconnect);
      socketInstance.off('connect_error', onConnectError);
      socketInstance.off('userId', onUserId);
      socketInstance.disconnect();
    };
  }, [serverUrl]);

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

export default SocketContext;
```

frontend\src\hooks\useWebRTCts

to top

```
import { useState, useEffect, useCallback, useRef } from 'react';
import { Socket } from 'socket.io-client';
import WebRTCService from '../services/WebRTCService';

interface UseWebRTCOptions {
  roomId?: string;
  onNewUser?: (userId: string) => void;
  onUserDisconnected?: (userId: string) => void;
  userName?: string;
}

const useWebRTC = (socket: Socket, options: UseWebRTCOptions) => {
  const { roomId, onNewUser, onUserDisconnected, userName } = options;
  const webRTCServiceRef = useRef<WebRTCService | null>(null);
  const [localStream, setLocalStream] = useState<MediaStream | null>(null);
  const [isConnected, setIsConnected] = useState(false);
  const [isVideoEnabled, setIsVideoEnabled] = useState(true);
  const [isAudioEnabled, setIsAudioEnabled] = useState(true);
  const [availableCameras, setAvailableCameras] = useState<MediaDeviceInfo[]>([]);
  const [currentCameraId, setCurrentCameraId] = useState<string | null>(null);
  const [error, setError] = useState<string | null>(null);
  const connectedUsersRef = useRef<Set<string>>(new Set<string>());
  const socketConnectedRef = useRef<boolean>(false);
  const [remoteStreams, setRemoteStreams] = useState<Map<string, MediaStream>>(new
Map());

  // Initialize media devices and stream
  const initializeMedia = useCallback(async () => {
    try {
      if (!webRTCServiceRef.current) {
        webRTCServiceRef.current = new WebRTCService(socket);
      }

      await webRTCServiceRef.current.setupMediaStream({ video: true, audio: true });
      setLocalStream(webRTCServiceRef.current.getLocalStream());
      setIsVideoEnabled(true);
      setIsAudioEnabled(true);

      // Get available cameras
      const devices = await webRTCServiceRef.current.getAvailableCameras();
      setAvailableCameras(devices);

      // Set current camera ID
      const videoTrack = webRTCServiceRef.current.getLocalStream()?.getVideoTracks()
[0];
      if (videoTrack) {
        const settings = videoTrack.getSettings();
        setCurrentCameraId(settings.deviceId || null);
      }

      return true;
    } catch (err) {
```

```
      console.error('Failed to initialize media:', err);
      setError('Could not access camera or microphone. Please check permissions.');
      return false;
    }
  }, [socket]);

  // Handle remote stream added event
  const handleRemoteStreamAdded = useCallback((event: Event) => {
    const customEvent = event as CustomEvent<{ userId: string; stream: MediaStream }>;
    const { userId, stream } = customEvent.detail;

    console.log(`Remote stream added for user ${userId}`);

    setRemoteStreams(prev => {
      const updated = new Map(prev);
      updated.set(userId, stream);
      return updated;
    });
  }, []);

  // Handle WebRTC signaling events
  const handleOffer = useCallback(async (offer: RTCSessionDescriptionInit, senderId:
string) => {
    if (!webRTCServiceRef.current) return;
    try {
      await webRTCServiceRef.current.handleReceivedOffer(offer, senderId, socket);
    } catch (err) {
      console.error('Error handling offer:', err);
      setError('Failed to establish connection with peer.');
    }
  }, [socket]);

  const handleAnswer = useCallback(async (answer: RTCSessionDescriptionInit, senderId:
string) => {
    if (!webRTCServiceRef.current) return;
    try {
      await webRTCServiceRef.current.handleReceivedAnswer(answer, senderId);
    } catch (err) {
      console.error('Error handling answer:', err);
      setError('Failed to establish connection with peer.');
    }
  }, []);

  const handleIceCandidate = useCallback(async (candidate: RTCIceCandidateInit,
senderId: string) => {
    if (!webRTCServiceRef.current) return;
    try {
      await webRTCServiceRef.current.handleIceCandidate(candidate, senderId);
    } catch (err) {
      console.error('Error handling ICE candidate:', err);
    }
  }, []);
```

```
  // Define joinRoom as a useCallback function
  const joinRoom = useCallback(() => {
    if (!socket || !roomId) return;

    console.log(`Joining room ${roomId} with username ${userName}`);

    // Send user data with join-room event
    const userData = { userName, userId: socket.id };
    socket.emit('join-room', roomId, userData);
  }, [socket, roomId, userName]);

  // Initialize socket connection
  useEffect(() => {
    if (!socket || !roomId) return;

    // Set socket connection timeout
    const connectionTimeout = setTimeout(() => {
      if (!socketConnectedRef.current) {
        console.error('Socket connection timeout. Attempting to reconnect...');
        setError('Connection timeout. Attempting to reconnect...');

        // Force socket reconnection
        socket.disconnect();
        socket.connect();
      }
    }, 5000);

    // Setup socket event listeners
    const handleSocketConnect = () => {
      console.log('Socket connected');
      socketConnectedRef.current = true;
      setError(null);

      // Join room with user data when socket is connected
      joinRoom();
    };

    const handleSocketDisconnect = () => {
      console.log('Socket disconnected');
      socketConnectedRef.current = false;
      setError('Connection lost. Attempting to reconnect...');
    };

    const handleSocketError = (err: Error) => {
      console.error('Socket error:', err);
      setError(`Connection error: ${err.message}`);
    };

    const handleSocketReconnect = () => {
      console.log('Socket reconnected');
      socketConnectedRef.current = true;
      setError(null);
```

```
      // Rejoin room after reconnection
      joinRoom();
    };

    // Add socket event listeners
    socket.on('connect', handleSocketConnect);
    socket.on('disconnect', handleSocketDisconnect);
    socket.on('error', handleSocketError);
    socket.on('reconnect', handleSocketReconnect);

    // Initial join if socket is already connected
    if (socket.connected) {
      console.log('Socket already connected, joining room immediately');
      socketConnectedRef.current = true;
      joinRoom();
    }

    // Cleanup function
    return () => {
      clearTimeout(connectionTimeout);
      socket.off('connect', handleSocketConnect);
      socket.off('disconnect', handleSocketDisconnect);
      socket.off('error', handleSocketError);
      socket.off('reconnect', handleSocketReconnect);
    };
  }, [socket, roomId, joinRoom]);

  useEffect(() => {
    if (!socket || !roomId) return;

    // Join the room when socket is connected
    if (socket.connected) {
      joinRoom();
    } else {
      // If socket is not connected, wait for it to connect
      socket.on('connect', joinRoom);
    }

    // Handle reconnection
    socket.io.on('reconnect', () => {
      console.log('Socket reconnected, rejoining room...');
      joinRoom();
    });

    return () => {
      socket.off('connect', joinRoom);
      socket.io.off('reconnect', joinRoom);
    };
  }, [socket, roomId, joinRoom]);

  // Join room and set up connections
  useEffect(() => {
    if (!socket || !roomId || !webRTCServiceRef.current || !localStream) return;
```

```
    console.log('Setting up WebRTC connections for room:', roomId);

    // Register socket event handlers
    socket.on('offer', handleOffer);
    socket.on('answer', handleAnswer);
    socket.on('ice-candidate', handleIceCandidate);

    // Listen for remote streams
    window.addEventListener('remote-stream-added', handleRemoteStreamAdded);

    // Notify server that we're ready for calls
    setIsConnected(true);

    // Handle user connections
    socket.on('user-connected', (userId: string, userData: any) => {
      console.log('User connected:', userId, userData);

      if (userId !== socket.id) {
        connectedUsersRef.current.add(userId);

        // Create a peer connection for the new user
        if (webRTCServiceRef.current && localStream) {
          try {
            webRTCServiceRef.current.createPeerConnection(userId);
            webRTCServiceRef.current.handleUserConnected(userId);
          } catch (err) {
            console.error('Error creating peer connection:', err);
          }
        }

        if (onNewUser) {
          onNewUser(userId);
        }
      }
    });

    socket.on('user-disconnected', (userId: string) => {
      console.log('User disconnected:', userId);
      connectedUsersRef.current.delete(userId);

      // Remove the remote stream when user disconnects
      setRemoteStreams(prev => {
        const updated = new Map(prev);
        updated.delete(userId);
        return updated;
      });

      if (webRTCServiceRef.current) {
        webRTCServiceRef.current.handleUserDisconnected(userId);
      }

      if (onUserDisconnected) {
```

```
        onUserDisconnected(userId);
      }
    });

    socket.on('room-users', (users: string[]) => {
      console.log('Current room users:', users);
      // Handle the list of users already in the room
      if (users && users.length > 0) {
        users.forEach((userId: string) => {
          if (userId !== socket.id) {
            connectedUsersRef.current.add(userId);

            // Create peer connections for existing users
            if (webRTCServiceRef.current && localStream) {
              try {
                webRTCServiceRef.current.createPeerConnection(userId);
                webRTCServiceRef.current.handleUserConnected(userId);
              } catch (err) {
                console.error('Error creating peer connection for existing user:',
err);
              }
            }

            if (onNewUser) {
              onNewUser(userId);
            }
          }
        });
      }
    });

    // Update remote streams from WebRTCService periodically
    const remoteStreamsInterval = setInterval(() => {
      if (webRTCServiceRef.current) {
        const serviceStreams = webRTCServiceRef.current.getRemoteStreams();
        const streamsMap = new Map<string, MediaStream>();

        Object.entries(serviceStreams).forEach(([userId, stream]) => {
          streamsMap.set(userId, stream);
        });

        setRemoteStreams(streamsMap);
      }
    }, 1000);

    const cleanup = () => {
      console.log('Cleaning up WebRTC connections');
      socket.off('offer', handleOffer);
      socket.off('answer', handleAnswer);
      socket.off('ice-candidate', handleIceCandidate);
      socket.off('user-connected');
      socket.off('user-disconnected');
      socket.off('room-users');
```

```
      window.removeEventListener('remote-stream-added', handleRemoteStreamAdded);
      clearInterval(remoteStreamsInterval);

      if (webRTCServiceRef.current) {
        webRTCServiceRef.current.closeAllConnections();
      }
    };

    return cleanup;
  }, [socket, roomId, localStream, handleOffer, handleAnswer, handleIceCandidate,
handleRemoteStreamAdded, onNewUser, onUserDisconnected]);

  // Toggle video
  const toggleVideo = useCallback(async () => {
    if (!webRTCServiceRef.current) return isVideoEnabled;

    try {
      const enabled = await webRTCServiceRef.current.toggleVideo();
      setIsVideoEnabled(enabled);
      return enabled;
    } catch (err) {
      console.error('Failed to toggle video:', err);
      setError('Failed to toggle video');
      return isVideoEnabled;
    }
  }, [isVideoEnabled]);

  // Toggle audio
  const toggleAudio = useCallback(() => {
    if (!webRTCServiceRef.current) return isAudioEnabled;

    try {
      const enabled = webRTCServiceRef.current.toggleAudioTrack();
      setIsAudioEnabled(enabled);
      return enabled;
    } catch (err) {
      console.error('Failed to toggle audio:', err);
      setError('Failed to toggle audio');
      return isAudioEnabled;
    }
  }, [isAudioEnabled]);

  // Switch camera
  const switchCamera = useCallback(async (deviceId: string) => {
    if (!webRTCServiceRef.current) throw new Error('WebRTC not initialized');

    try {
      await webRTCServiceRef.current.switchCamera(deviceId);
      setCurrentCameraId(deviceId);
    } catch (err) {
      console.error('Failed to switch camera:', err);
      setError('Failed to switch camera');
      throw err;
```

```
    }
  }, []);

  // Leave meeting
  const leaveMeeting = useCallback(() => {
    if (socket && roomId) {
      socket.emit('leave-room', roomId);
    }

    if (webRTCServiceRef.current) {
      webRTCServiceRef.current.closeAllConnections();
      webRTCServiceRef.current.stopLocalStream();
    }

    setLocalStream(null);
    setIsConnected(false);
    connectedUsersRef.current.clear();
  }, [socket, roomId]);

  return {
    initializeMedia,
    toggleVideo,
    toggleAudio,
    switchCamera,
    leaveMeeting,
    localStream,
    isConnected,
    isVideoEnabled,
    isAudioEnabled,
    availableCameras,
    currentCameraId,
    error,
    remoteStreams
  };
};

export default useWebRTC;
```

frontend\src\maintsx

to top

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.tsx'

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

frontend\src\pages\Dashboardtsx

to top

```
import React, { useState } from 'react';
import JoinMeetingForm from '../components/video-call/JoinMeetingForm';
import { useSocket } from '../context/SocketContext';

// Tab types
type TabType = 'home' | 'calendar' | 'meetings';

const Dashboard: React.FC = () => {
  const { socket, isConnected } = useSocket();
  const [error, setError] = useState<string | null>(null);
  const [activeTab, setActiveTab] = useState<TabType>('home');

  const createMeeting = async (meetingName: string, userName: string): Promise<string>
=> {
    return new Promise((resolve, reject) => {
      if (!socket || !isConnected) {
        const errorMessage = 'Socket connection not available';
        setError(errorMessage);
        reject(new Error(errorMessage));
        return;
      }

      try {
        // Generate a random meeting ID
        const meetingId = Math.random().toString(36).substring(2, 10);

        // In a real implementation, you would create the meeting on the server
        // and include the meetingName and userName in the request
        console.log(`Creating meeting "${meetingName}" for user "${userName}"`);

        // For now, we'll just resolve with the generated ID
        setTimeout(() => resolve(meetingId), 500);
      } catch (err) {
        const errorMsg = err instanceof Error ? err.message : 'Unknown error creating
meeting';
        setError(errorMsg);
        reject(new Error(errorMsg));
      }
    });
  };

  const renderTabContent = () => {
    switch (activeTab) {
      case 'home':
        return (
          <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
            <div className="bg-white overflow-hidden shadow rounded-lg">
              <div className="px-4 py-5 sm:p-6">
                <h2 className="text-lg font-medium text-gray-900 mb-4">Welcome to
Workoutmate</h2>
                <p className="text-gray-600 mb-6">
                  Your fitness companion for scheduling and joining workout sessions
```

```
with friends and trainers.
                </p>
                <div className="grid grid-cols-2 gap-4">
                  <div className="bg-blue-50 p-4 rounded-lg">
                    <h3 className="font-medium text-blue-800 mb-2">My Workout
Stats</h3>
                    <p className="text-blue-700">No workouts completed yet.</p>
                  </div>
                  <div className="bg-green-50 p-4 rounded-lg">
                    <h3 className="font-medium text-green-800 mb-2">Upcoming
Sessions</h3>
                    <p className="text-green-700">No upcoming sessions.</p>
                  </div>
                </div>
              </div>
            </div>
            <div className="bg-white overflow-hidden shadow rounded-lg">
              <div className="px-4 py-5 sm:p-6">
                <h2 className="text-lg font-medium text-gray-900 mb-4">Recent
Activity</h2>
                <p className="text-gray-600 mb-6">
                  Your recent workout activities and messages will appear here.
                </p>
                <div className="border border-gray-200 rounded-lg p-4 bg-gray-50">
                  <p className="text-gray-500 text-center">No recent activity</p>
                </div>
              </div>
            </div>
          </div>
        );
      case 'calendar':
        return (
          <div className="bg-white overflow-hidden shadow rounded-lg">
            <div className="px-4 py-5 sm:p-6">
              <h2 className="text-lg font-medium text-gray-900 mb-4">Workout
Calendar</h2>
              <p className="text-gray-600 mb-6">
                View and manage your scheduled workout sessions.
              </p>
              <div className="border border-gray-200 rounded-lg p-4 bg-gray-50 h-96
flex items-center justify-center">
                <p className="text-gray-500 text-center">
                  Calendar component will be implemented here.<br />
                  <span className="text-sm text-blue-500 mt-2 block">Coming from
workoutmate-convex2 integration</span>
                </p>
              </div>
            </div>
          </div>
        );
      case 'meetings':
        return (
          <div className="bg-white overflow-hidden shadow rounded-lg">
```

```
            <div className="px-4 py-5 sm:p-6">
              <h2 className="text-lg font-medium text-gray-900 mb-4">Video
Meetings</h2>
              <p className="text-gray-600 mb-6">
                Create or join a video meeting for your workout session. You can
connect with your
                workout partner or trainer in real-time.
              </p>
              <JoinMeetingForm onCreateMeeting={createMeeting} />

              <div className="mt-8 border-t pt-6">
                <h3 className="text-md font-medium text-gray-900 mb-4">Recent
Meetings</h3>
                <div className="border border-gray-200 rounded-lg p-4 bg-gray-50">
                  <p className="text-gray-500 text-center">No recent meetings</p>
                </div>
              </div>
            </div>
          </div>
        );
      default:
        return null;
    }
  };

  return (
    <div className="min-h-screen bg-gray-100">
      <header className="bg-white shadow">
        <div className="max-w-7xl mx-auto px-4 py-6 sm:px-6 lg:px-8">
          <h1 className="text-3xl font-bold text-gray-900">Workoutmate</h1>
        </div>
      </header>
      <main>
        <div className="max-w-7xl mx-auto py-6 sm:px-6 lg:px-8">
          {/* Status indicator */}
          <div className="mb-4 flex items-center">
            <div className={`w-3 h-3 rounded-full mr-2 ${isConnected ? 'bg-green-500' :
'bg-red-500'}`}></div>
            <p className="text-sm text-gray-600">
              {isConnected ? 'Connected to server' : 'Disconnected from server'}
            </p>
          </div>

          {error && (
            <div className="mb-6 p-4 bg-red-100 border border-red-200 text-red-700
rounded">
              {error}
            </div>
          )}

          {/* Tabs */}
          <div className="border-b border-gray-200 mb-6">
            <nav className="-mb-px flex space-x-8">
```

```
              <button
                onClick={() => setActiveTab('home')}
                className={`whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm
${
                  activeTab === 'home'
                    ? 'border-blue-500 text-blue-600'
                    : 'border-transparent text-gray-500 hover:text-gray-700
hover:border-gray-300'
                }`}
              >
                Home
              </button>
              <button
                onClick={() => setActiveTab('calendar')}
                className={`whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm
${
                  activeTab === 'calendar'
                    ? 'border-blue-500 text-blue-600'
                    : 'border-transparent text-gray-500 hover:text-gray-700
hover:border-gray-300'
                }`}
              >
                Calendar
              </button>
              <button
                onClick={() => setActiveTab('meetings')}
                className={`whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm
${
                  activeTab === 'meetings'
                    ? 'border-blue-500 text-blue-600'
                    : 'border-transparent text-gray-500 hover:text-gray-700
hover:border-gray-300'
                }`}
              >
                Meetings
              </button>
            </nav>
          </div>

          {/* Tab content */}
          {renderTabContent()}
        </div>
      </main>
    </div>
  );
};

export default Dashboard;
```

frontend\src\pages\Meetingtsx

to top

```tsx
import React, { useEffect, useState } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { useSocket } from '../context/SocketContext';
import VideoCallRoom from '../components/video-call/VideoCallRoom';

const Meeting: React.FC = () => {
  const { roomId } = useParams<{ roomId: string }>();
  const navigate = useNavigate();
  const { socket, isConnected } = useSocket();
  const [userName, setUserName] = useState<string>('');
  const [error, setError] = useState<string | null>(null);
  const [isJoiningRoom, setIsJoiningRoom] = useState<boolean>(false);

  useEffect(() => {
    // Retrieve username from localStorage
    const storedUserName = localStorage.getItem('userName');
    if (storedUserName) {
      setUserName(storedUserName);
    }

    // Redirect if no roomId is provided
    if (!roomId) {
      navigate('/dashboard');
      return;
    }

    // Check if socket is connected
    if (!isConnected) {
      setError('Not connected to server. Please try again later.');
      return;
    }

    setIsJoiningRoom(true);

    // Cleanup function
    return () => {
      if (socket && roomId) {
        socket.emit('leave-room', roomId);
        setIsJoiningRoom(false);
      }
    };
  }, [socket, isConnected, roomId, navigate]);

  const handleLeaveCall = () => {
    navigate('/dashboard');
  };

  if (error) {
    return (
      <div className="min-h-screen bg-gray-100 flex flex-col items-center justify-
center p-4">
        <div className="bg-white p-6 rounded-lg shadow-md w-full max-w-md text-center">
```

```
          <h2 className="text-xl font-semibold text-red-600 mb-4">Error</h2>
          <p className="text-gray-700 mb-6">{error}</p>
          <button
            onClick={() => navigate('/dashboard')}
            className="px-4 py-2 bg-blue-600 text-white rounded hover:bg-blue-700
focus:outline-none focus:ring-2 focus:ring-blue-500 focus:ring-offset-2"
          >
            Return to Dashboard
          </button>
        </div>
      </div>
    );
  }

  if (!isConnected || !socket) {
    return (
      <div className="min-h-screen bg-gray-100 flex flex-col items-center justify-
center p-4">
        <div className="bg-white p-6 rounded-lg shadow-md w-full max-w-md text-center">
          <h2 className="text-xl font-semibold text-gray-800 mb-4">Connecting...</h2>
          <p className="text-gray-600 mb-6">Trying to establish a connection to the
server.</p>
          <div className="animate-pulse flex justify-center">
            <div className="h-3 w-3 bg-blue-600 rounded-full mr-1"></div>
            <div className="h-3 w-3 bg-blue-600 rounded-full mr-1 animate-pulse delay-
150"></div>
            <div className="h-3 w-3 bg-blue-600 rounded-full animate-pulse delay-300">
</div>
          </div>
        </div>
      </div>
    );
  }

  return (
    <div className="min-h-screen bg-gray-100">
      <header className="bg-white shadow">
        <div className="max-w-7xl mx-auto px-4 py-4 sm:px-6 lg:px-8 flex justify-
between items-center">
          <h1 className="text-2xl font-bold text-gray-900">Meeting: {roomId}</h1>
          <button
            onClick={handleLeaveCall}
            className="px-3 py-1 bg-red-600 text-white text-sm rounded hover:bg-red-700
focus:outline-none focus:ring-2 focus:ring-red-500 focus:ring-offset-2"
          >
            Leave Meeting
          </button>
        </div>
      </header>
      <main className="max-w-7xl mx-auto py-6 sm:px-6 lg:px-8">
        <div className="px-4 py-6 sm:px-0">
          <div className="border-4 border-dashed border-gray-200 rounded-lg min-h-
[70vh] p-4">
```

```
            {isJoiningRoom && socket && (
              <VideoCallRoom
                socket={socket}
                userName={userName}
              />
            )}
          </div>
        </div>
      </main>
    </div>
  );
};

export default Meeting;
```

frontend\src\services\WebRTCServicets

to top

```
import { Socket } from 'socket.io-client';

export interface PeerConnection {
  connection: RTCPeerConnection;
  videoElement?: HTMLDivElement;
}

export interface VideoTrackSender {
  track?: MediaStreamTrack;
  kind?: string;
  replaceTrack: (track: MediaStreamTrack) => Promise<void>;
}

class WebRTCService {
  private socket: Socket;
  private peers: Record<string, RTCPeerConnection> = {};
  private peerVideoElements: Record<string, HTMLDivElement> = {};
  private localStream: MediaStream | null = null;
  private currentCamera: string | null = null;
  // Track connection states
  private connectionStates: Record<string, string> = {};
  // Buffer for ICE candidates that arrive before peer connection is established
  private iceCandidateBuffers: Record<string, RTCIceCandidateInit[]> = {};
  // Track remote streams
  private remoteStreams: Record<string, MediaStream> = {};
  // Connection attempt timestamps to prevent rapid reconnection attempts
  private connectionAttemptTimestamps: Record<string, number> = {};
  // Minimum time between connection attempts in ms
  private readonly MIN_RECONNECTION_INTERVAL = 3000;

  constructor(socket: Socket) {
    this.socket = socket;
  }

  async setupMediaStream(constraints: MediaStreamConstraints = { video: true, audio:
true }): Promise<MediaStream> {
    try {
      this.localStream = await navigator.mediaDevices.getUserMedia(constraints);
      return this.localStream;
    } catch (error) {
      console.error('Media stream error:', error);
      throw new Error('Error accessing camera and microphone');
    }
  }

  getLocalStream(): MediaStream | null {
    return this.localStream;
  }

  createPeerConnection(userId: string): RTCPeerConnection {
    // Check if we've attempted to connect recently to prevent rapid reconnection
attempts
```

```
    const now = Date.now();
    const lastAttempt = this.connectionAttemptTimestamps[userId] || 0;
    if (now - lastAttempt < this.MIN_RECONNECTION_INTERVAL) {
      console.log(`Skipping reconnection attempt to ${userId} - too soon since last
attempt`);
      if (this.peers[userId]) {
        return this.peers[userId];
      }
    }

    // Update connection attempt timestamp
    this.connectionAttemptTimestamps[userId] = now;

    // Close existing connection if it exists
    if (this.peers[userId]) {
      try {
        this.peers[userId].close();
      } catch (e) {
        console.warn('Error closing existing peer connection:', e);
      }
    }

    console.log(`Creating peer connection for user ${userId}`);

    const peerConnection = new RTCPeerConnection({
      iceServers: [
        { urls: 'stun:stun.l.google.com:19302' },
        { urls: 'stun:stun1.l.google.com:19302' },
        { urls: 'stun:stun2.l.google.com:19302' }
      ],
      iceCandidatePoolSize: 10
    });

    // Initialize connection state
    this.connectionStates[userId] = 'new';

    // Log connection state changes
    peerConnection.onconnectionstatechange = () => {
      console.log(`Connection state for ${userId}: ${peerConnection.connectionState}`);
      this.connectionStates[userId] = peerConnection.connectionState;

      // Handle connection failures
      if (peerConnection.connectionState === 'failed' || peerConnection.connectionState
=== 'disconnected') {
        console.warn(`Connection to ${userId} ${peerConnection.connectionState},
attempting recovery...`);

        // Don't immediately try to reconnect - let the useWebRTC hook handle
reconnection
        // This prevents cascading reconnection attempts
      }
    };
```

```
    // Log signaling state changes
    peerConnection.onsignalingstatechange = () => {
      console.log(`Signaling state for ${userId}: ${peerConnection.signalingState}`);
    };

    // Handle ICE connection state changes
    peerConnection.oniceconnectionstatechange = () => {
      console.log(`ICE connection state for ${userId}:
${peerConnection.iceConnectionState}`);

      // If ICE connection fails, we might need to restart ICE
      if (peerConnection.iceConnectionState === 'failed') {
        console.warn(`ICE connection failed for ${userId}, will attempt to restart
ICE`);
        // We'll handle ICE restart when needed through renegotiation
      }
    };

    // Add local tracks to the peer connection
    if (this.localStream) {
      this.localStream.getTracks().forEach(track => {
        if (this.localStream) {
          console.log(`Adding ${track.kind} track to peer connection for ${userId}`);
          peerConnection.addTrack(track, this.localStream);
        }
      });
    }

    peerConnection.onicecandidate = event => {
      if (event.candidate) {
        console.log(`Sending ICE candidate to ${userId}`);
        this.socket.emit('ice-candidate', event.candidate, userId);
      }
    };

    peerConnection.ontrack = event => {
      console.log(`Received track from ${userId}:`, event.track.kind);

      if (event.streams && event.streams[0]) {
        const stream = event.streams[0];
        console.log(`Stream ID: ${stream.id}, Track ID: ${event.track.id}, Kind:
${event.track.kind}`);

        // Store the stream for this peer
        this.remoteStreams[userId] = stream;

        // Always dispatch the event with the complete stream
        // This ensures the UI gets updated even if we already have a video element
        const customEvent = new CustomEvent('remote-stream-added', {
          detail: { userId, stream }
        });

        // Use a small timeout to ensure all tracks are added before dispatching
```

```
        setTimeout(() => {
          window.dispatchEvent(customEvent);
          console.log(`Dispatched remote-stream-added event for ${userId} with stream
ID ${stream.id}`);
        }, 100);

        if (!this.peerVideoElements[userId]) {
          // Create video element for this peer
          const videoElement = this.createVideoElement(stream, false);
          this.peerVideoElements[userId] = videoElement;

          // Add user ID to the video element for debugging
          videoElement.setAttribute('data-user-id', userId);

          console.log(`Created video element for user ${userId}`);
        } else {
          // Update existing video element with new stream
          const videoElement = this.peerVideoElements[userId];
          const videoChild = videoElement.querySelector('video');
          if (videoChild) {
            videoChild.srcObject = stream;
            console.log(`Updated video element for user ${userId} with new stream`);
          }
        }
      } else {
        console.warn(`Received track from ${userId} but no stream was provided`);

        // If no stream was provided, create one from the track
        if (!this.remoteStreams[userId]) {
          this.remoteStreams[userId] = new MediaStream();
        }

        // Add the track to the stream
        const stream = this.remoteStreams[userId];
        stream.addTrack(event.track);

        console.log(`Added track ${event.track.id} (${event.track.kind}) to manually
created stream for ${userId}`);

        // Dispatch event after adding the track
        const customEvent = new CustomEvent('remote-stream-added', {
          detail: { userId, stream }
        });

        // Use a small timeout to ensure all tracks are added before dispatching
        setTimeout(() => {
          window.dispatchEvent(customEvent);
          console.log(`Dispatched remote-stream-added event for ${userId} with manually
created stream`);
        }, 100);
      }
    };
```

```
    this.peers[userId] = peerConnection;

    // Process any buffered ICE candidates for this peer
    this.processBufferedIceCandidates(userId);

    return peerConnection;
  }

  // Process any buffered ICE candidates for this peer
  async processBufferedIceCandidates(userId: string): Promise<void> {
    const peerConnection = this.peers[userId];
    const bufferedCandidates = this.iceCandidateBuffers[userId] || [];

    if (peerConnection && bufferedCandidates.length > 0) {
      console.log(`Processing ${bufferedCandidates.length} buffered ICE candidates for
${userId}`);

      for (const candidate of bufferedCandidates) {
        try {
          await peerConnection.addIceCandidate(new RTCIceCandidate(candidate));
          console.log(`Added buffered ICE candidate for ${userId}`);
        } catch (error) {
          console.error(`Error adding buffered ICE candidate for ${userId}:`, error);
        }
      }

      // Clear the buffer
      this.iceCandidateBuffers[userId] = [];
    }
  }

  createVideoElement(stream: MediaStream, isLocal: boolean): HTMLDivElement {
    // Create container
    const container = document.createElement('div');
    container.className = 'relative bg-black rounded-lg overflow-hidden';

    // Create video element
    const video = document.createElement('video');
    video.srcObject = stream;
    video.autoplay = true;
    if (isLocal) {
      video.muted = true; // Mute local video to prevent feedback
    }
    video.className = 'w-full h-full object-cover';

    // Add label
    const label = document.createElement('div');
    label.className = 'absolute bottom-2 left-2 bg-black bg-opacity-60 text-white px-2
py-1 text-xs rounded';
    label.textContent = isLocal ? 'You' : 'Remote User';

    container.appendChild(video);
    container.appendChild(label);
```

```
    return container;
  }

  async connectToNewUser(userId: string): Promise<RTCPeerConnection> {
    console.log(`Connecting to new user: ${userId}`);

    // Create peer connection if it doesn't exist
    const peerConnection = this.createPeerConnection(userId);

    try {
      // Create offer
      const offer = await peerConnection.createOffer();
      await peerConnection.setLocalDescription(offer);

      console.log(`Sending offer to ${userId}`);
      this.socket.emit('offer', offer, userId);

      return peerConnection;
    } catch (error) {
      console.error(`Error connecting to user ${userId}:`, error);
      throw new Error(`Failed to connect to user ${userId}`);
    }
  }

  async handleReceivedOffer(offer: RTCSessionDescriptionInit, senderId: string, socket:
Socket): Promise<void> {
    console.log(`Received offer from ${senderId}`);

    try {
      // Create peer connection if it doesn't exist
      const peerConnection = this.createPeerConnection(senderId);

      // Check if we can set remote description
      const currentState = peerConnection.signalingState;
      if (currentState !== 'stable') {
        console.warn(`Signaling state is ${currentState}, not stable. Proceeding with
caution.`);

        // If we have a local description and we're in have-local-offer state,
        // we need to rollback before setting the remote offer
        if (currentState === 'have-local-offer') {
          console.log('Rolling back local description to handle remote offer');
          await peerConnection.setLocalDescription({type: 'rollback'});
        }
      }

      // Set remote description
      await peerConnection.setRemoteDescription(new RTCSessionDescription(offer));

      // Create answer
      const answer = await peerConnection.createAnswer();
      await peerConnection.setLocalDescription(answer);
```

```
      console.log(`Sending answer to ${senderId}`);
      socket.emit('answer', answer, senderId);
    } catch (error) {
      console.error(`Error handling offer from ${senderId}:`, error);
      throw new Error(`Failed to handle offer from ${senderId}`);
    }
  }

  async handleReceivedAnswer(answer: RTCSessionDescriptionInit, senderId: string):
Promise<void> {
    console.log(`Received answer from ${senderId}`);

    const peerConnection = this.peers[senderId];
    if (!peerConnection) {
      console.warn(`No peer connection found for ${senderId}`);
      return;
    }

    try {
      const currentState = peerConnection.signalingState;

      // We should only apply the answer if we're in have-local-offer state
      if (currentState !== 'have-local-offer') {
        console.warn(`Unexpected signaling state ${currentState} when receiving answer.
Expected 'have-local-offer'.`);

        // If we're in stable state, we might have already processed this answer
        if (currentState === 'stable') {
          console.log('Already in stable state, ignoring duplicate answer');
          return;
        }
      }

      await peerConnection.setRemoteDescription(new RTCSessionDescription(answer));
      console.log(`Successfully set remote description for ${senderId}`);
    } catch (error) {
      console.error(`Error handling answer from ${senderId}:`, error);
      throw new Error(`Failed to handle answer from ${senderId}`);
    }
  }

  async handleIceCandidate(candidate: RTCIceCandidateInit, senderId: string):
Promise<void> {
    console.log(`Received ICE candidate from ${senderId}`);

    const peerConnection = this.peers[senderId];

    // If we don't have a peer connection yet, buffer the candidate
    if (!peerConnection) {
      console.log(`No peer connection for ${senderId} yet, buffering ICE candidate`);

      if (!this.iceCandidateBuffers[senderId]) {
```

```
        this.iceCandidateBuffers[senderId] = [];
      }

      this.iceCandidateBuffers[senderId].push(candidate);
      return;
    }

    // If the connection isn't ready to receive candidates, buffer them
    if (peerConnection.remoteDescription === null) {
      console.log(`Remote description not set for ${senderId}, buffering ICE
candidate`);

      if (!this.iceCandidateBuffers[senderId]) {
        this.iceCandidateBuffers[senderId] = [];
      }

      this.iceCandidateBuffers[senderId].push(candidate);
      return;
    }

    try {
      await peerConnection.addIceCandidate(new RTCIceCandidate(candidate));
      console.log(`Added ICE candidate for ${senderId}`);
    } catch (error) {
      console.error(`Error adding ICE candidate for ${senderId}:`, error);
      throw new Error(`Failed to add ICE candidate for ${senderId}`);
    }
  }

  handleUserConnected(userId: string): void {
    console.log(`User connected: ${userId}`);
    this.connectToNewUser(userId).catch(error => {
      console.error(`Failed to connect to user ${userId}:`, error);
    });
  }

  handleUserDisconnected(userId: string): void {
    this.closePeerConnection(userId);
  }

  closePeerConnection(userId: string): void {
    console.log(`Closing peer connection for ${userId}`);

    // Close and remove peer connection
    if (this.peers[userId]) {
      try {
        this.peers[userId].close();
      } catch (e) {
        console.warn(`Error closing peer connection for ${userId}:`, e);
      }

      delete this.peers[userId];
    }
```

```
    // Remove video element
    if (this.peerVideoElements[userId]) {
      delete this.peerVideoElements[userId];
    }

    // Remove remote stream
    if (this.remoteStreams[userId]) {
      delete this.remoteStreams[userId];
    }

    // Clear connection state
    delete this.connectionStates[userId];

    // Clear buffered ICE candidates
    delete this.iceCandidateBuffers[userId];

    // Clear connection attempt timestamp
    delete this.connectionAttemptTimestamps[userId];
  }

  closeAllConnections(): void {
    console.log('Closing all peer connections');
    Object.keys(this.peers).forEach(userId => {
      this.closePeerConnection(userId);
    });
  }

  async toggleVideo(): Promise<boolean> {
    if (!this.localStream) return false;

    const videoTrack = this.localStream.getVideoTracks()[0];
    if (videoTrack) {
      if (videoTrack.enabled) {
        // If video is currently enabled, just disable it
        videoTrack.enabled = false;
        return false;
      } else {
        // If video is currently disabled, enable it
        videoTrack.enabled = true;
        return true;
      }
    }
    return false;
  }

  toggleAudioTrack(): boolean {
    if (!this.localStream) return false;

    const audioTrack = this.localStream.getAudioTracks()[0];
    if (audioTrack) {
      audioTrack.enabled = !audioTrack.enabled;
      return audioTrack.enabled;
```

```
    }
    return false;
  }

  stopLocalStream(): void {
    if (this.localStream) {
      this.localStream.getTracks().forEach(track => track.stop());
      this.localStream = null;
    }
  }

  async getAvailableCameras(): Promise<MediaDeviceInfo[]> {
    try {
      const devices = await navigator.mediaDevices.enumerateDevices();
      return devices.filter(device => device.kind === 'videoinput');
    } catch (error) {
      console.error('Error getting cameras:', error);
      return [];
    }
  }

  async switchCamera(deviceId: string): Promise<boolean> {
    try {
      // Create a new stream with just the video from the new camera
      const newStream = await navigator.mediaDevices.getUserMedia({
        video: { deviceId: { exact: deviceId } },
        audio: false
      });

      if (!this.localStream) {
        // If we don't have a local stream yet, create one with audio
        const audioStream = await navigator.mediaDevices.getUserMedia({ audio: true });
        this.localStream = new MediaStream();

        // Add the video track from the new camera
        newStream.getVideoTracks().forEach(track => {
          this.localStream!.addTrack(track);
        });

        // Add the audio track
        audioStream.getAudioTracks().forEach(track => {
          this.localStream!.addTrack(track);
        });
      } else {
        // If we have an existing stream, replace just the video track
        const oldVideoTracks = this.localStream.getVideoTracks();

        // Remove old video tracks
        oldVideoTracks.forEach(track => {
          track.stop();
          this.localStream!.removeTrack(track);
        });
```

```
        // Add the new video track
        newStream.getVideoTracks().forEach(track => {
          this.localStream!.addTrack(track);
        });

        // Replace the track in all peer connections
        Object.values(this.peers).forEach(peer => {
          const senders = peer.getSenders();
          const videoSender = senders.find(sender =>
            sender.track?.kind === 'video'
          );

          if (videoSender && this.localStream) {
            const videoTrack = this.localStream.getVideoTracks()[0];
            if (videoTrack) {
              videoSender.replaceTrack(videoTrack).catch(error => {
                console.error('Error replacing track:', error);
              });
            }
          }
        });
      }

      this.currentCamera = deviceId;
      return true;
    } catch (error) {
      console.error('Error switching camera:', error);
      throw new Error('Failed to switch camera');
    }
  }

  // Methods for React components to access video elements
  getLocalVideoElement(): HTMLDivElement | null {
    if (!this.localStream) return null;
    return this.createVideoElement(this.localStream, true);
  }

  getPeerVideoElements(): HTMLDivElement[] {
    return Object.values(this.peerVideoElements);
  }

  // Get peer connection by user ID
  getPeerConnection(userId: string): RTCPeerConnection | undefined {
    return this.peers[userId];
  }

  // Get all peer connections
  getAllPeerConnections(): Record<string, RTCPeerConnection> {
    return this.peers;
  }

  // Get current camera ID
  getCurrentCamera(): string | null {
```

```
    return this.currentCamera;
  }

  // Get all remote streams
  getRemoteStreams(): Record<string, MediaStream> {
    return this.remoteStreams;
  }

  // Check if a user is connected
  isUserConnected(userId: string): boolean {
    return this.peers[userId] !== undefined &&
           this.connectionStates[userId] === 'connected';
  }

  // Restart ICE for a specific peer connection
  async restartIceForPeer(userId: string): Promise<void> {
    const peerConnection = this.peers[userId];
    if (!peerConnection) {
      console.warn(`No peer connection found for ${userId}, cannot restart ICE`);
      return;
    }

    try {
      // Create a new offer with ICE restart flag
      const offer = await peerConnection.createOffer({ iceRestart: true });
      await peerConnection.setLocalDescription(offer);

      console.log(`Sending new offer with ICE restart to ${userId}`);
      this.socket.emit('offer', offer, userId);
    } catch (error) {
      console.error(`Error restarting ICE for ${userId}:`, error);
      throw new Error(`Failed to restart ICE for ${userId}`);
    }
  }
}

export default WebRTCService;
```

frontend\src\vite-envd.ts

to top

```
/// <reference types="vite/client" />
```

frontend\tailwindconfig.js

to top

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

frontend\viteconfig.ts

to top

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vite.dev/config/
export default defineConfig({
  plugins: [react()],
})
```

READMEmd

to top

Workoutmate

A fitness application for planning and joining workout sessions with your mate, built with Node.js, Express, and Socket.IO, following Domain-Driven Design principles.

Features

- User authentication (Local & Google OAuth)
- Real-time workout session tracking
- Workout planning and scheduling
- Session joining with workout mates
- Secure session management

Prerequisites

- Node.js (v18 or higher)
- npm (v8 or higher)

Setup

1. Clone the repository:

```
git clone <repository-url>
cd workoutmate
```

2. Install dependencies:

```
npm install
```

3. Set up environment variables:

```
cp .env.example .env
```

Then edit `.env` with your configuration:

- Set your `SESSION_SECRET`
- Configure Google OAuth credentials
- Adjust ports if needed

4. Start the development server:

```
npm run dev
```

The application will be available at `http://localhost:3000`

Project Structure

The project follows Domain-Driven Design principles:

```
src/
├── domain/          # Domain entities, value objects, and interfaces
├── application/     # Application services and use cases
├── infrastructure/  # External services, repositories, and technical concerns
├── interfaces/      # API routes, controllers, and presenters
└── shared/         # Shared utilities and constants
```

Development

- `npm start`: Start the server with nodemon
- `npm run dev`: Start the server and tunnel for local development

- `npm run tunnel` : Start localtunnel only

Environment Variables

- `PORT` : Server port (default: 3000)
- `NODE_ENV` : Environment (development/production)
- `SESSION_SECRET` : Secret for session encryption
- `GOOGLE_CLIENT_ID` : Google OAuth client ID
- `GOOGLE_CLIENT_SECRET` : Google OAuth client secret
- `WS_PORT` : WebSocket port (if different from main port)

Contributing

1. Create a feature branch
2. Commit your changes
3. Push to the branch
4. Create a Pull Request

License

MIT (LICENSE)

src\app\pagetsx

to top

```
export default async function Home() {
  const meetings = await prisma.meeting.findMany({
    orderBy: {
      date: 'asc'
    }
  })

  return (
    <main className="container mx-auto p-4">
      <h1 className="text-2xl font-bold mb-6">Meeting Scheduler</h1>
      <div className="space-y-8">
        <CreateMeetingForm />
        <div>
          <h2 className="text-xl font-semibold mb-4">Available Meetings</h2>
          <MeetingList meetings={meetings} />
        </div>
      </div>
    </main>
  )
}
```

src\application\middlewares\asyncmiddleware.js

to top

```
/**
 * Async handler middleware to avoid try-catch blocks in controllers
 * Wraps controller functions in a try-catch block
 * @param {Function} fn - The async function to execute
 * @returns {Function} Middleware function
 */
const asyncHandler = fn => (req, res, next) => {
  Promise.resolve(fn(req, res, next)).catch(next);
};

module.exports = asyncHandler;
```

src\application\middlewares\authmiddleware.js

to top

```
/**
 * Authentication middleware for API routes
 */

/**
 * Middleware to ensure user is authenticated
 * @param {Object} req - Express request object
 * @param {Object} res - Express response object
 * @param {Function} next - Express next function
 */
const isAuthenticated = (req, res, next) => {
    if (req.isAuthenticated()) {
        return next();
    }
    return res.status(401).json({ error: 'Unauthorized. Please log in to continue.' });
};

/**
 * Middleware to validate user is the owner or participant of a resource
 * @param {Object} req - Express request object
 * @param {Object} res - Express response object
 * @param {Function} next - Express next function
 */
const isOwnerOrParticipant = (model) => async (req, res, next) => {
    try {
        const resourceId = req.params.id;
        const resource = await model.findById(resourceId);

        if (!resource) {
            return res.status(404).json({ error: 'Resource not found' });
        }

        // Check if user is the creator/owner
        if (resource.creator && resource.creator.toString() === req.user.id) {
            return next();
        }

        // Check if user is a participant
        if (resource.participants && resource.participants.some(p => p.user &&
p.user.toString() === req.user.id)) {
            return next();
        }

        return res.status(403).json({ error: 'Forbidden. You do not have permission to
access this resource.' });
    } catch (error) {
        return res.status(500).json({ error: 'Server error' });
    }
};

module.exports = {
    isAuthenticated,
```

```
    isOwnerOrParticipant
};
```

src\application\services\AuthServicejs

to top

```
const User = require('@domain/entities/User');
const {
    UserAlreadyExistsException,
    InvalidCredentialsException
} = require('@domain/shared/exceptions/DomainException');
const { UserLoggedIn } = require('@domain/events/UserEvents');
const DomainEventPublisher = require('@domain/events/DomainEventPublisher');

class AuthService {
    constructor(userRepository) {
        this.userRepository = userRepository;
    }

    async registerUser(email, password, name) {
        const existingUser = await this.userRepository.findByEmail(email);
        if (existingUser) {
            throw new UserAlreadyExistsException(email);
        }

        const user = User.createNew(email, name);
        await user.setPassword(password);

        return this.userRepository.save(user);
    }

    async authenticateUser(email, password) {
        const user = await this.userRepository.findByEmail(email);
        if (!user) {
            throw new InvalidCredentialsException();
        }

        const isValid = await user.comparePassword(password);
        if (!isValid) {
            throw new InvalidCredentialsException();
        }

        // Publish login event
        DomainEventPublisher.getInstance().publish(
            new UserLoggedIn(user.id, 'local')
        );

        return user;
    }

    async authenticateGoogleUser(profile) {
        let user = await this.userRepository.findByGoogleId(profile.id);

        if (!user) {
            // Check if user exists with same email
            user = await this.userRepository.findByEmail(profile.emails[0].value);

            if (user) {
```

```
                // Link Google account to existing user
                return this.userRepository.update(user.id, {
                    googleId: profile.id,
                    profilePicture: profile.photos[0].value
                });
            } else {
                // Create new user
                const newUser = User.createFromGoogle(
                    profile.emails[0].value,
                    profile.displayName,
                    profile.id,
                    profile.photos[0].value
                );
                return this.userRepository.save(newUser);
            }
        }

        // Publish login event
        DomainEventPublisher.getInstance().publish(
            new UserLoggedIn(user.id, 'google')
        );

        return user;
    }

    async getUserById(id) {
        return this.userRepository.findById(id);
    }
}

module.exports = AuthService;
```

src\application\services\ChatServicejs

to top

```
const Chat = require('../../domain/aggregates/Chat');
const { v4: uuidv4 } = require('uuid');

class ChatService {
    constructor(chatRepository, meetingService) {
        this.chatRepository = chatRepository;
        this.meetingService = meetingService;
        // In-memory storage for chat messages
        this.chatRooms = new Map();
    }

    async initializeChat(roomId) {
        if (!this.chatRooms.has(roomId)) {
            this.chatRooms.set(roomId, []);
        }
    }

    async sendMessage(roomId, senderId, content) {
        const message = {
            id: Date.now().toString(),
            roomId,
            senderId,
            content,
            timestamp: new Date(),
            toJSON() {
                return {
                    id: this.id,
                    senderId: this.senderId,
                    content: this.content,
                    timestamp: this.timestamp
                };
            }
        };

        const room = this.chatRooms.get(roomId) || [];
        room.push(message);
        this.chatRooms.set(roomId, room);

        return message;
    }

    async getMessages(roomId, limit = 50, before = new Date()) {
        const room = this.chatRooms.get(roomId) || [];
        return room
            .filter(msg => msg.timestamp < before)
            .slice(-limit)
            .reverse();
    }

    async getMessage(messageId) {
        const message = await this.chatRepository.getMessage(messageId);
        if (!message) {
```

```
            throw new Error('Message not found');
        }
        return message;
    }
}

module.exports = ChatService;
```

src\application\services\MeetingServicejs

to top

```
const { v4: uuidv4 } = require('uuid');
const Meeting = require('../../domain/entities/Meeting');

class MeetingService {
    constructor(meetingRepository) {
        this.meetingRepository = meetingRepository;
    }

    createMeeting() {
        const meeting = new Meeting(uuidv4());
        return this.meetingRepository.create(meeting);
    }

    createMeetingWithId(id) {
        const meeting = new Meeting(id);
        return this.meetingRepository.create(meeting);
    }

    getMeeting(id) {
        return this.meetingRepository.findById(id);
    }

    addParticipant(meetingId, participantId) {
        let meeting = this.meetingRepository.findById(meetingId);
        if (!meeting) {
            // Create the meeting if it doesn't exist
            console.log(`Meeting ${meetingId} not found, creating it automatically`);
            meeting = new Meeting(meetingId);
            this.meetingRepository.create(meeting);
        }
        meeting.addParticipant(participantId);
        return this.meetingRepository.update(meeting);
    }

    removeParticipant(meetingId, participantId) {
        let meeting = this.meetingRepository.findById(meetingId);
        if (!meeting) {
            // Just log a warning instead of throwing an error
            console.warn(`Attempted to remove participant ${participantId} from non-
existent meeting ${meetingId}`);
            return null;
        }
        meeting.removeParticipant(participantId);
        return this.meetingRepository.update(meeting);
    }
}

module.exports = MeetingService;
```

src\components\CreateMeetingFormtsx

to top

```tsx
import { useRouter } from 'next/navigation'

export function CreateMeetingForm() {
  const router = useRouter()

  const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault()
    // ... existing form submission code ...

    try {
      const response = await fetch('/api/meetings', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(formData),
      })

      if (response.ok) {
        router.refresh()
        form.reset()
      } else {
        console.error('Failed to create meeting')
      }
    } catch (error) {
      console.error('Error creating meeting:', error)
    }
  }

  // ... rest of the component code ...
}
```

src\components\MeetingListtsx

to top

```tsx
import { Meeting } from '@/types/Meeting'

interface MeetingListProps {
  meetings: Meeting[]
}

export function MeetingList({ meetings }: MeetingListProps) {
  return (
    <div className="space-y-4">
      {meetings.map((meeting) => (
        <div
          key={meeting._id}
          className="p-4 bg-white rounded-lg shadow"
        >
          <h3 className="text-lg font-semibold">{meeting.title}</h3>
          <p className="text-gray-600">{meeting.description}</p>
          <div className="mt-2 text-sm text-gray-500">
            <p>Date: {new Date(meeting.date).toLocaleDateString()}</p>
            <p>Time: {meeting.time}</p>
            <p>Duration: {meeting.duration} minutes</p>
          </div>
        </div>
      ))}
    </div>
  )
}
```

src\domain\aggregates\Chatjs

to top

```
const Message = require('../entities/Message');

class Chat {
    constructor(meetingId) {
        this.meetingId = meetingId;
        this.messages = [];
    }

    addMessage(messageId, senderId, content) {
        const message = new Message(messageId, this.meetingId, senderId, content);
        this.messages.push(message);
        return message;
    }

    getMessages(limit = 50, before = new Date()) {
        return this.messages
            .filter(message => message.getTimestamp() < before)
            .sort((a, b) => b.getTimestamp() - a.getTimestamp())
            .slice(0, limit);
    }

    getMessageById(messageId) {
        return this.messages.find(message => message.getId() === messageId);
    }

    getMeetingId() {
        return this.meetingId;
    }

    toJSON() {
        return {
            meetingId: this.meetingId,
            messages: this.messages.map(message => message.toJSON())
        };
    }
}

module.exports = Chat;
```

src\domain\entities\Meetingjs

to top

```
class Meeting {
    constructor(id, createdAt = new Date()) {
        this.id = id;
        this.createdAt = createdAt;
        this.participants = new Set();
    }

    addParticipant(participantId) {
        this.participants.add(participantId);
    }

    removeParticipant(participantId) {
        this.participants.delete(participantId);
    }

    hasParticipant(participantId) {
        return this.participants.has(participantId);
    }

    getParticipants() {
        return Array.from(this.participants);
    }
}

module.exports = Meeting;
```

src\domain\entities\Messagejs

to top

```
class Message {
    constructor(id, meetingId, senderId, content, timestamp = new Date()) {
        this.id = id;
        this.meetingId = meetingId;
        this.senderId = senderId;
        this.content = content;
        this.timestamp = timestamp;
    }

    getId() {
        return this.id;
    }

    getMeetingId() {
        return this.meetingId;
    }

    getSenderId() {
        return this.senderId;
    }

    getContent() {
        return this.content;
    }

    getTimestamp() {
        return this.timestamp;
    }

    toJSON() {
        return {
            id: this.id,
            meetingId: this.meetingId,
            senderId: this.senderId,
            content: this.content,
            timestamp: this.timestamp
        };
    }
}

module.exports = Message;
```

src\domain\entities\Userjs

to top

```
const bcrypt = require('bcryptjs');
const Email = require('@domain/valueObjects/Email');
const DomainEventPublisher = require('@domain/events/DomainEventPublisher');
const { UserRegistered, GoogleAccountLinked, ProfileUpdated } =
require('@domain/events/UserEvents');

class User {
    #id;
    #email;
    #name;
    #password;
    #googleId;
    #profilePicture;
    #createdAt;

    constructor(id, email, name, password = null, googleId = null, profilePicture =
null) {
        this.#id = id;
        this.#email = new Email(email);
        this.#name = name;
        this.#password = password;
        this.#googleId = googleId;
        this.#profilePicture = profilePicture;
        this.#createdAt = new Date();
    }

    // Getters
    get id() { return this.#id; }
    get email() { return this.#email.value; }
    get name() { return this.#name; }
    get googleId() { return this.#googleId; }
    get profilePicture() { return this.#profilePicture; }
    get createdAt() { return this.#createdAt; }

    // Domain methods
    async setPassword(plainPassword) {
        if (!plainPassword) {
            throw new Error('Password cannot be empty');
        }
        this.#password = await bcrypt.hash(plainPassword, 10);
    }

    async comparePassword(candidatePassword) {
        if (!this.#password) return false;
        return bcrypt.compare(candidatePassword, this.#password);
    }

    linkGoogleAccount(googleId, profilePicture) {
        if (this.#googleId) {
            throw new Error('Google account already linked');
        }
        this.#googleId = googleId;
```

```
        this.#profilePicture = profilePicture;

        // Publish domain event
        DomainEventPublisher.getInstance().publish(
            new GoogleAccountLinked(this.#id, googleId)
        );
    }

    updateProfile(name, profilePicture) {
        const changes = {};
        if (name && name !== this.#name) {
            this.#name = name;
            changes.name = name;
        }
        if (profilePicture && profilePicture !== this.#profilePicture) {
            this.#profilePicture = profilePicture;
            changes.profilePicture = profilePicture;
        }

        if (Object.keys(changes).length > 0) {
            // Publish domain event
            DomainEventPublisher.getInstance().publish(
                new ProfileUpdated(this.#id, changes)
            );
        }
    }

    toJSON() {
        return {
            id: this.#id,
            email: this.#email.value,
            name: this.#name,
            profilePicture: this.#profilePicture,
            createdAt: this.#createdAt
        };
    }

    // Factory methods
    static createNew(email, name) {
        const user = new User(null, email, name);

        // Publish domain event
        DomainEventPublisher.getInstance().publish(
            new UserRegistered(user.id, email, name, 'local')
        );

        return user;
    }

    static createFromGoogle(email, name, googleId, profilePicture) {
        const user = new User(null, email, name, null, googleId, profilePicture);

        // Publish domain event
```

```
        DomainEventPublisher.getInstance().publish(
            new UserRegistered(user.id, email, name, 'google')
        );

        return user;
    }
}

module.exports = User;
```

src\domain\events\DomainEventPublisherjs

to top

```
class DomainEventPublisher {
    static #instance;
    #handlers;

    constructor() {
        if (DomainEventPublisher.#instance) {
            return DomainEventPublisher.#instance;
        }
        this.#handlers = new Map();
        DomainEventPublisher.#instance = this;
    }

    static getInstance() {
        if (!DomainEventPublisher.#instance) {
            DomainEventPublisher.#instance = new DomainEventPublisher();
        }
        return DomainEventPublisher.#instance;
    }

    subscribe(eventType, handler) {
        if (!this.#handlers.has(eventType)) {
            this.#handlers.set(eventType, new Set());
        }
        this.#handlers.get(eventType).add(handler);
    }

    unsubscribe(eventType, handler) {
        if (this.#handlers.has(eventType)) {
            this.#handlers.get(eventType).delete(handler);
        }
    }

    async publish(event) {
        const eventType = event.constructor.name;
        if (this.#handlers.has(eventType)) {
            const handlers = this.#handlers.get(eventType);
            const promises = Array.from(handlers).map(handler => handler(event));
            await Promise.all(promises);
        }
    }
}

module.exports = DomainEventPublisher;
```

src\domain\events\UserEventsjs

to top

```
class UserRegistered {
    constructor(userId, email, name, registrationType) {
        this.userId = userId;
        this.email = email;
        this.name = name;
        this.registrationType = registrationType; // 'local' or 'google'
        this.timestamp = new Date();
    }
}

class UserLoggedIn {
    constructor(userId, loginType) {
        this.userId = userId;
        this.loginType = loginType; // 'local' or 'google'
        this.timestamp = new Date();
    }
}

class GoogleAccountLinked {
    constructor(userId, googleId) {
        this.userId = userId;
        this.googleId = googleId;
        this.timestamp = new Date();
    }
}

class ProfileUpdated {
    constructor(userId, changes) {
        this.userId = userId;
        this.changes = changes;
        this.timestamp = new Date();
    }
}

module.exports = {
    UserRegistered,
    UserLoggedIn,
    GoogleAccountLinked,
    ProfileUpdated
};
```

src\domain\models\eventmodel.js

to top

```
const mongoose = require('mongoose');

const eventSchema = new mongoose.Schema({
  title: {
    type: String,
    required: [true, 'Event title is required'],
    trim: true
  },
  description: {
    type: String,
    trim: true
  },
  start: {
    type: Date,
    required: [true, 'Start date/time is required']
  },
  end: {
    type: Date,
    required: [true, 'End date/time is required']
  },
  allDay: {
    type: Boolean,
    default: false
  },
  creator: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: [true, 'Creator is required']
  },
  participants: [{
    user: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    },
    status: {
      type: String,
      enum: ['pending', 'accepted', 'declined'],
      default: 'pending'
    }
  }],
  location: {
    type: String,
    trim: true
  },
  color: {
    type: String,
    default: '#3788d8'
  },
  isWorkoutSession: {
    type: Boolean,
    default: false
  },
```

```
  sessionDetails: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'WorkoutSession'
  },
  createdAt: {
    type: Date,
    default: Date.now
  },
  updatedAt: {
    type: Date,
    default: Date.now
  }
});

// Update timestamp on save
eventSchema.pre('save', function(next) {
  this.updatedAt = Date.now();
  next();
});

module.exports = mongoose.model('Event', eventSchema);
```

src\domain\models\messagemodel.js

to top

```
const mongoose = require('mongoose');

const messageSchema = new mongoose.Schema({
  sender: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: [true, 'Sender is required']
  },
  content: {
    type: String,
    required: [true, 'Message content is required'],
    trim: true
  },
  eventId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Event',
    required: [true, 'Event ID is required']
  },
  readBy: [{
    user: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    },
    readAt: {
      type: Date,
      default: Date.now
    }
  }],
  createdAt: {
    type: Date,
    default: Date.now
  }
}, {
  timestamps: true
});

// Index to improve query performance for event chats
messageSchema.index({ eventId: 1, createdAt: 1 });

module.exports = mongoose.model('Message', messageSchema);
```

src\domain\models\workout-sessionmodel.js

to top

```
const mongoose = require('mongoose');

const workoutSessionSchema = new mongoose.Schema({
  title: {
    type: String,
    required: [true, 'Session title is required'],
    trim: true
  },
  description: {
    type: String,
    trim: true
  },
  eventId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Event',
    required: [true, 'Associated event is required']
  },
  workoutType: {
    type: String,
    enum: ['cardio', 'strength', 'flexibility', 'hiit', 'yoga', 'other'],
    default: 'other'
  },
  intensity: {
    type: String,
    enum: ['low', 'medium', 'high'],
    default: 'medium'
  },
  exercises: [{
    name: {
      type: String,
      required: true
    },
    sets: {
      type: Number
    },
    reps: {
      type: Number
    },
    duration: {
      type: Number // in minutes
    },
    notes: {
      type: String
    }
  }],
  equipment: [{
    type: String
  }],
  goals: {
    type: String
  },
  maxParticipants: {
```

```
    type: Number,
    default: 5
  },
  isPrivate: {
    type: Boolean,
    default: false
  },
  videoCallEnabled: {
    type: Boolean,
    default: true
  },
  createdAt: {
    type: Date,
    default: Date.now
  },
  updatedAt: {
    type: Date,
    default: Date.now
  }
});

// Update timestamp on save
workoutSessionSchema.pre('save', function(next) {
  this.updatedAt = Date.now();
  next();
});

module.exports = mongoose.model('WorkoutSession', workoutSessionSchema);
```

src\domain\repositories\ChatRepositoryjs

to top

```
class ChatRepository {
    async createChat(meetingId) {
        throw new Error('Method not implemented');
    }

    async getChat(meetingId) {
        throw new Error('Method not implemented');
    }

    async saveMessage(message) {
        throw new Error('Method not implemented');
    }

    async getMessages(meetingId, limit = 50, before = new Date()) {
        throw new Error('Method not implemented');
    }

    async getMessage(messageId) {
        throw new Error('Method not implemented');
    }
}

module.exports = ChatRepository;
```

src\domain\repositories\IMeetingRepositoryjs

to top

```
class IMeetingRepository {
    create(meeting) {
        throw new Error('Method not implemented');
    }

    findById(id) {
        throw new Error('Method not implemented');
    }

    update(meeting) {
        throw new Error('Method not implemented');
    }

    delete(id) {
        throw new Error('Method not implemented');
    }
}

module.exports = IMeetingRepository;
```

src\domain\repositories\IUserRepositoryjs

to top

```
/**
 * @interface IUserRepository
 * Repository interface for User entity operations
 */
class IUserRepository {
    /**
     * Find a user by their ID
     * @param {string} id
     * @returns {Promise<import('../entities/User')>}
     */
    async findById(id) {
        throw new Error('Method not implemented');
    }

    /**
     * Find a user by their email
     * @param {string} email
     * @returns {Promise<import('../entities/User')>}
     */
    async findByEmail(email) {
        throw new Error('Method not implemented');
    }

    /**
     * Find a user by their Google ID
     * @param {string} googleId
     * @returns {Promise<import('../entities/User')>}
     */
    async findByGoogleId(googleId) {
        throw new Error('Method not implemented');
    }

    /**
     * Save a user
     * @param {import('../entities/User')} user
     * @returns {Promise<import('../entities/User')>}
     */
    async save(user) {
        throw new Error('Method not implemented');
    }

    /**
     * Update a user
     * @param {string} id
     * @param {Partial<import('../entities/User')>} userData
     * @returns {Promise<import('../entities/User')>}
     */
    async update(id, userData) {
        throw new Error('Method not implemented');
    }

    /**
```

```
     * Delete a user
     * @param {string} id
     * @returns {Promise<boolean>}
     */
    async delete(id) {
        throw new Error('Method not implemented');
    }
}

module.exports = IUserRepository;
```

src\domain\shared\exceptions\DomainExceptionjs

to top

```
class DomainException extends Error {
    constructor(message) {
        super(message);
        this.name = this.constructor.name;
    }
}

class InvalidEmailException extends DomainException {
    constructor(email) {
        super(`Invalid email format: ${email}`);
        this.email = email;
    }
}

class UserAlreadyExistsException extends DomainException {
    constructor(email) {
        super(`User already exists with email: ${email}`);
        this.email = email;
    }
}

class InvalidCredentialsException extends DomainException {
    constructor() {
        super('Invalid email or password');
    }
}

class GoogleAccountAlreadyLinkedException extends DomainException {
    constructor(userId) {
        super('Google account already linked to this user');
        this.userId = userId;
    }
}

module.exports = {
    DomainException,
    InvalidEmailException,
    UserAlreadyExistsException,
    InvalidCredentialsException,
    GoogleAccountAlreadyLinkedException
};
```

src\domain\valueObjects\Emailjs

to top

```
class Email {
    #value;

    constructor(email) {
        this.validate(email);
        this.#value = email.toLowerCase();
    }

    validate(email) {
        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (!email || !emailRegex.test(email)) {
            throw new Error('Invalid email format');
        }
    }

    equals(other) {
        return other instanceof Email && this.#value === other.value;
    }

    get value() {
        return this.#value;
    }

    toString() {
        return this.#value;
    }
}

module.exports = Email;
```

src\indexjs

to top

```
require('module-alias/register');
require('dotenv').config();
const express = require('express');
const session = require('express-session');
const bodyParser = require('body-parser');
const path = require('path');
const http = require('http');
const socketIo = require('socket.io');
const mongoose = require('mongoose');

// Infrastructure
const InMemoryMeetingRepository =
require('@infrastructure/persistence/InMemoryMeetingRepository');
const InMemoryUserRepository =
require('@infrastructure/persistence/InMemoryUserRepository');
const AuthenticationProvider = require('@infrastructure/auth/AuthenticationProvider');
const WebSocketService = require('@infrastructure/websocket/WebSocketService');

// Application Services
const MeetingService = require('@application/services/MeetingService');
const AuthService = require('@application/services/AuthService');
const ChatService = require('@application/services/ChatService');

// Interface Routes
const createMeetingRoutes = require('@interfaces/http/routes/meetingRoutes');
const createAuthRoutes = require('@interfaces/http/routes/authRoutes');
const { ensureAuth } = require('@infrastructure/auth/middleware/auth');

// New routes
const eventRoutes = require('./interfaces/routes/event.routes');
const messageRoutes = require('./interfaces/routes/message.routes');

// Initialize Express app and server
const app = express();
const server = http.createServer(app);
const io = socketIo(server, {
  cors: {
    origin: process.env.FRONTEND_URL || "http://localhost:3000",
    methods: ["GET", "POST"],
    credentials: true
  }
});

// Basic middleware
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// Session configuration
app.use(session({
    secret: process.env.SESSION_SECRET || 'your-super-secret-session-key',
    resave: false,
    saveUninitialized: false,
```

```
    cookie: {
        secure: process.env.NODE_ENV === 'production',
        maxAge: 24 * 60 * 60 * 1000 // 24 hours
    }
}));

// Connect to MongoDB
mongoose.connect(process.env.MONGODB_URI || 'mongodb://localhost:27017/workoutmate', {
    useNewUrlParser: true,
    useUnifiedTopology: true
})
.then(() => console.log('MongoDB Connected'))
.catch(err => console.log('MongoDB Connection Error:', err));

// Initialize repositories
const meetingRepository = new InMemoryMeetingRepository();
const userRepository = new InMemoryUserRepository();

// Initialize services
const meetingService = new MeetingService(meetingRepository);
const authService = new AuthService(userRepository);
const chatService = new ChatService();

// Initialize authentication provider
const authProvider = new AuthenticationProvider(authService);
const { initialize, session: passportSession } = authProvider.getMiddleware();
app.use(initialize);
app.use(passportSession);

// Initialize WebSocket service
const webSocketService = new WebSocketService(io, meetingService, chatService);

// Make Socket.IO instance available to routes
app.set('io', io);

// Serve public assets (images, css, etc.)
app.use('/assets', express.static(path.join(__dirname, 'public')));

// Serve auth-specific static assets (available to all)
app.use('/static/auth', express.static(path.join(__dirname, 'interfaces/web/auth')));

// Auth routes (must be before protected routes)
app.use('/auth', createAuthRoutes(authService, authProvider));

// Protected API routes
app.use('/api', ensureAuth, createMeetingRoutes(meetingService));

// New API routes
app.use('/api/events', eventRoutes);
app.use('/api/messages', messageRoutes);

// Protected routes
app.get('/dashboard', ensureAuth, (req, res) => {
```

```
    res.sendFile(path.join(__dirname, 'interfaces/web/dashboard/pages/index.html'));
});

app.get('/meeting/:id', ensureAuth, async (req, res) => {
    try {
        const meeting = await meetingService.getMeeting(req.params.id);
        if (!meeting) {
            res.redirect('/dashboard?error=meeting-not-found');
            return;
        }
        res.sendFile(path.join(__dirname,
'interfaces/web/dashboard/pages/index.html'));
    } catch (error) {
        res.redirect('/dashboard?error=invalid-meeting');
    }
});

// Calendar route
app.get('/calendar', ensureAuth, (req, res) => {
    res.sendFile(path.join(__dirname, 'interfaces/web/dashboard/pages/index.html'));
});

// Public routes
app.get('/', (req, res) => {
    if (req.isAuthenticated()) {
        res.redirect('/dashboard');
    } else {
        res.redirect('/login');
    }
});

app.get('/login', (req, res) => {
    if (req.isAuthenticated()) {
        res.redirect('/dashboard');
    } else {
        res.sendFile(path.join(__dirname, 'interfaces/web/auth/pages/login.html'));
    }
});

app.get('/signup', (req, res) => {
    if (req.isAuthenticated()) {
        res.redirect('/dashboard');
    } else {
        res.sendFile(path.join(__dirname, 'interfaces/web/auth/pages/signup.html'));
    }
});

// Protected interface assets (js, services, etc.)
app.use('/static', ensureAuth, express.static(path.join(__dirname, 'interfaces/web')));

// Handle Socket.IO events for messaging and WebRTC
io.on('connection', (socket) => {
    // Store the authenticated user on the socket
```

```
    if (socket.request.user && socket.request.user.logged_in) {
        socket.user = socket.request.user;
        console.log(`Socket connected for user: ${socket.user.username ||
socket.user.email}`);

        // Join user to their personal room
        socket.join(`user:${socket.user._id}`);

        // Handle joining event rooms (for chat & video)
        socket.on('join-event', (eventId) => {
            socket.join(`event:${eventId}`);
            // Notify others in the room
            socket.to(`event:${eventId}`).emit('user-joined', {
                userId: socket.user._id,
                username: socket.user.username || socket.user.email
            });
        });

        // Handle leaving event rooms
        socket.on('leave-event', (eventId) => {
            socket.leave(`event:${eventId}`);
            // Notify others in the room
            socket.to(`event:${eventId}`).emit('user-left', {
                userId: socket.user._id
            });
        });

        // WebRTC signaling
        socket.on('offer', (offer, toUserId) => {
            socket.to(`user:${toUserId}`).emit('offer', offer, socket.user._id);
        });

        socket.on('answer', (answer, toUserId) => {
            socket.to(`user:${toUserId}`).emit('answer', answer, socket.user._id);
        });

        socket.on('ice-candidate', (candidate, toUserId) => {
            socket.to(`user:${toUserId}`).emit('ice-candidate', candidate,
socket.user._id);
        });

        // Handle disconnect
        socket.on('disconnect', () => {
            console.log(`Socket disconnected for user: ${socket.user.username ||
socket.user.email}`);
        });
    }
});

// Start server
const port = process.env.PORT || 3000;
server.listen(port, () => {
```

```
    console.log(`Server running at http://localhost:${port}`);
});
```

src\infrastructure\auth\AuthenticationProviderjs

to top

```
const passport = require('passport');
const createLocalStrategy = require('./strategies/LocalStrategy');
const createGoogleStrategy = require('./strategies/GoogleStrategy');
const configureUserSerialization = require('./serialization/UserSerialization');

class AuthenticationProvider {
    constructor(authService) {
        this.passport = passport;
        this.authService = authService;
        this.initialize();
    }

    initialize() {
        // Configure strategies
        this.passport.use(createLocalStrategy(this.authService));
        this.passport.use(createGoogleStrategy(this.authService));

        // Configure serialization
        configureUserSerialization(this.passport, this.authService);
    }

    getPassportInstance() {
        return this.passport;
    }

    getMiddleware() {
        return {
            initialize: this.passport.initialize(),
            session: this.passport.session()
        };
    }

    authenticate(strategy, options = {}) {
        return this.passport.authenticate(strategy, options);
    }
}

module.exports = AuthenticationProvider;
```

src\infrastructure\auth\middleware\authjs

to top

```
const ensureAuth = (req, res, next) => {
    if (req.isAuthenticated()) {
        return next();
    }
    // Redirect to login page for HTML requests, send 401 for API requests
    if (req.accepts('html')) {
        res.redirect('/login');
    } else {
        res.status(401).json({ error: 'Please log in to continue' });
    }
};

const ensureGuest = (req, res, next) => {
    if (!req.isAuthenticated()) {
        return next();
    }
    res.redirect('/dashboard');
};

module.exports = {
    ensureAuth,
    ensureGuest
};
```

src\infrastructure\auth\passportConfigjs

to top

```
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;
const GoogleStrategy = require('passport-google-oauth20').Strategy;

function configurePassport(authService) {
    // Local Strategy
    passport.use(new LocalStrategy({
        usernameField: 'email',
        passwordField: 'password'
    }, async (email, password, done) => {
        try {
            const user = await authService.authenticateUser(email, password);
            return done(null, user);
        } catch (error) {
            return done(null, false, { message: error.message });
        }
    }));

    // Google Strategy
    passport.use(new GoogleStrategy({
        clientID: process.env.GOOGLE_CLIENT_ID,
        clientSecret: process.env.GOOGLE_CLIENT_SECRET,
        callbackURL: '/auth/google/callback'
    }, async (accessToken, refreshToken, profile, done) => {
        try {
            const user = await authService.authenticateGoogleUser(profile);
            return done(null, user);
        } catch (error) {
            return done(error);
        }
    }));

    // Serialize user for the session
    passport.serializeUser((user, done) => {
        done(null, user.id);
    });

    // Deserialize user from the session
    passport.deserializeUser(async (id, done) => {
        try {
            const user = await authService.getUserById(id);
            done(null, user);
        } catch (error) {
            done(error);
        }
    });

    return passport;
}

module.exports = configurePassport;
```

src\infrastructure\auth\serialization\UserSerializationjs

to top

```
function configureUserSerialization(passport, authService) {
    passport.serializeUser((user, done) => {
        done(null, user.id);
    });

    passport.deserializeUser(async (id, done) => {
        try {
            const user = await authService.getUserById(id);
            done(null, user);
        } catch (error) {
            done(error);
        }
    });
}

module.exports = configureUserSerialization;
```

src\infrastructure\auth\strategies\GoogleStrategyjs

to top

```
const GoogleStrategy = require('passport-google-oauth20').Strategy;

function createGoogleStrategy(authService) {
    return new GoogleStrategy({
        clientID: process.env.GOOGLE_CLIENT_ID,
        clientSecret: process.env.GOOGLE_CLIENT_SECRET,
        callbackURL: '/auth/google/callback'
    }, async (accessToken, refreshToken, profile, done) => {
        try {
            const user = await authService.authenticateGoogleUser(profile);
            return done(null, user);
        } catch (error) {
            return done(error);
        }
    });
}

module.exports = createGoogleStrategy;
```

src\infrastructure\auth\strategies\LocalStrategyjs

to top

```
const LocalStrategy = require('passport-local').Strategy;

function createLocalStrategy(authService) {
    return new LocalStrategy({
        usernameField: 'email',
        passwordField: 'password'
    }, async (email, password, done) => {
        try {
            const user = await authService.authenticateUser(email, password);
            return done(null, user);
        } catch (error) {
            return done(null, false, { message: error.message });
        }
    });
}

module.exports = createLocalStrategy;
```

src\infrastructure\persistence\InMemoryChatRepositoryjs

to top

```
const ChatRepository = require('../../domain/repositories/ChatRepository');
const Chat = require('../../domain/aggregates/Chat');

class InMemoryChatRepository extends ChatRepository {
    constructor() {
        super();
        this.chats = new Map();
        this.messages = new Map();
    }

    async createChat(meetingId) {
        if (this.chats.has(meetingId)) {
            throw new Error('Chat already exists for this meeting');
        }
        const chat = new Chat(meetingId);
        this.chats.set(meetingId, chat);
        this.messages.set(meetingId, []);
        return chat;
    }

    async getChat(meetingId) {
        return this.chats.get(meetingId);
    }

    async saveMessage(message) {
        const meetingMessages = this.messages.get(message.getMeetingId()) || [];
        meetingMessages.push(message);
        this.messages.set(message.getMeetingId(), meetingMessages);
        return message;
    }

    async getMessages(meetingId, limit = 50, before = new Date()) {
        const meetingMessages = this.messages.get(meetingId) || [];
        return meetingMessages
            .filter(message => message.getTimestamp() < before)
            .sort((a, b) => b.getTimestamp() - a.getTimestamp())
            .slice(0, limit);
    }

    async getMessage(messageId) {
        for (const messages of this.messages.values()) {
            const message = messages.find(m => m.getId() === messageId);
            if (message) return message;
        }
        return null;
    }
}

module.exports = InMemoryChatRepository;
```

src\infrastructure\persistence\InMemoryMeetingRepositoryjs

to top

```
const IMeetingRepository = require('../../domain/repositories/IMeetingRepository');

class InMemoryMeetingRepository extends IMeetingRepository {
    constructor() {
        super();
        this.meetings = new Map();
    }

    create(meeting) {
        this.meetings.set(meeting.id, meeting);
        return meeting;
    }

    findById(id) {
        return this.meetings.get(id) || null;
    }

    update(meeting) {
        if (!this.meetings.has(meeting.id)) {
            throw new Error('Meeting not found');
        }
        this.meetings.set(meeting.id, meeting);
        return meeting;
    }

    delete(id) {
        this.meetings.delete(id);
    }
}

module.exports = InMemoryMeetingRepository;
```

src\infrastructure\persistence\InMemoryUserRepositoryjs

to top

```
const { v4: uuidv4 } = require('uuid');
const User = require('@domain/entities/User');
const IUserRepository = require('@domain/repositories/IUserRepository');

class InMemoryUserRepository extends IUserRepository {
    constructor() {
        super();
        this.users = new Map();
    }

    async findById(id) {
        return this.users.get(id) || null;
    }

    async findByEmail(email) {
        return Array.from(this.users.values()).find(user => user.email === email) ||
null;
    }

    async findByGoogleId(googleId) {
        return Array.from(this.users.values()).find(user => user.googleId === googleId)
|| null;
    }

    async save(userData) {
        const id = userData.id || uuidv4();
        let user;

        if (userData instanceof User) {
            user = userData;
            Object.defineProperty(user, 'id', { value: id });
        } else {
            user = new User(
                id,
                userData.email,
                userData.name,
                userData.password,
                userData.googleId,
                userData.profilePicture
            );
        }

        this.users.set(id, user);
        return user;
    }

    async update(id, userData) {
        const user = this.users.get(id);
        if (!user) return null;

        if (userData.name || userData.profilePicture) {
            user.updateProfile(userData.name, userData.profilePicture);
```

```
        }

        if (userData.googleId) {
            user.linkGoogleAccount(userData.googleId, userData.profilePicture);
        }

        return user;
    }

    async delete(id) {
        return this.users.delete(id);
    }
}

module.exports = InMemoryUserRepository;
```

src\infrastructure\websocket\WebSocketServicejs

to top

```
class WebSocketService {
    constructor(io, meetingService, chatService) {
        this.io = io;
        this.meetingService = meetingService;
        this.chatService = chatService;
        this.setupSocketHandlers();
    }

    setupSocketHandlers() {
        this.io.on('connection', (socket) => {
            console.log('User connected:', socket.id);

            // Increase max listeners to avoid warnings
            socket.setMaxListeners(20);

            // Track which room this socket is in
            let currentRoom = null;

            // Handle create meeting request
            socket.on('create-meeting', async ({ meetingId }, callback) => {
                try {
                    // Check if meeting exists
                    let meeting = await this.meetingService.getMeeting(meetingId);

                    // If meeting doesn't exist, create it
                    if (!meeting) {
                        // In the original implementation, this would create a random
ID
                        // But we're using the ID from the frontend instead
                        meeting = await
this.meetingService.createMeetingWithId(meetingId);
                    }

                    callback({ success: true, meetingId });
                } catch (error) {
                    console.error('Error creating meeting:', error);
                    callback({ success: false, error: error.message });
                }
            });

            // Join room handler
            socket.on('join-room', (roomId, userData) => {
                console.log(`User ${socket.id} joining room ${roomId}`, userData);

                try {
                    // Leave previous room if any
                    if (currentRoom) {
                        socket.leave(currentRoom);
                        try {
                            this.meetingService.removeParticipant(currentRoom,
socket.id);
```

```
                            // Get updated participants list after removal
                            const previousMeeting =
this.meetingService.getMeeting(currentRoom);
                            const previousParticipants = previousMeeting ?
previousMeeting.participants : [];

                            // Emit updated participants list to the previous room
                            this.io.to(currentRoom).emit('room-users',
previousParticipants);

                            socket.to(currentRoom).emit('user-disconnected',
socket.id);
                        } catch (error) {
                            console.warn(`Error removing participant from previous room
${currentRoom}:`, error.message);
                        }
                    }

                    // Store the current room ID
                    currentRoom = roomId;

                    // Join the new room
                    socket.join(roomId);

                    // Add participant to the meeting - this will create the meeting if
it doesn't exist
                    this.meetingService.addParticipant(roomId, socket.id);

                    // Get all participants in the room
                    const meeting = this.meetingService.getMeeting(roomId);
                    const participants = meeting ? meeting.participants : [];

                    // Log participants for debugging
                    console.log(`Room ${roomId} participants:`, participants);

                    // Emit to everyone in the room except the sender
                    socket.to(roomId).emit('user-connected', socket.id, userData);

                    // Send the current participants list to the new user
                    socket.emit('room-users', participants);

                    // Also send the updated participants list to everyone in the room
                    this.io.to(roomId).emit('room-users', participants);

                    console.log(`User ${socket.id} connected to room ${roomId}. Total
participants: ${participants.length}`);

                    // Initialize chat for the meeting
                    this.chatService.initializeChat(roomId).catch(err => {
                        console.warn(`Error initializing chat for room ${roomId}:`,
err.message);
                    });
                } catch (error) {
```

```
                    console.error(`Error joining room ${roomId}:`, error);
                    socket.emit('error', { message: 'Failed to join meeting room',
details: error.message });
                }
            });

            // Chat events
            socket.on('chat-message', async (content) => {
                if (!currentRoom) return;

                try {
                    const message = await this.chatService.sendMessage(currentRoom,
socket.id, content);
                    this.io.to(currentRoom).emit('chat-message', message.toJSON());
                } catch (error) {
                    socket.emit('chat-error', error.message);
                }
            });

            socket.on('get-chat-history', async (before) => {
                if (!currentRoom) return;

                try {
                    const messages = await this.chatService.getMessages(currentRoom,
50, new Date(before));
                    socket.emit('chat-history', messages.map(m => m.toJSON()));
                } catch (error) {
                    socket.emit('chat-error', error.message);
                }
            });

            // WebRTC signaling
            socket.on('offer', (offer, recipientId) => {
                socket.to(recipientId).emit('offer', offer, socket.id);
            });

            socket.on('answer', (answer, recipientId) => {
                socket.to(recipientId).emit('answer', answer, socket.id);
            });

            socket.on('ice-candidate', (candidate, recipientId) => {
                socket.to(recipientId).emit('ice-candidate', candidate, socket.id);
            });

            // Disconnect handler
            socket.on('disconnect', () => {
                console.log('User disconnected:', socket.id);
                if (currentRoom) {
                    try {
                        this.meetingService.removeParticipant(currentRoom, socket.id);

                        // Get updated participants list after removal
                        const meeting = this.meetingService.getMeeting(currentRoom);
```

```
                        const participants = meeting ? meeting.participants : [];

                        // Emit updated participants list to everyone in the room
                        this.io.to(currentRoom).emit('room-users', participants);

                        socket.to(currentRoom).emit('user-disconnected', socket.id);

                        console.log(`User ${socket.id} disconnected from room
${currentRoom}. Remaining participants: ${participants.length}`);
                    } catch (error) {
                        console.warn(`Error removing participant ${socket.id} from room
${currentRoom} on disconnect:`, error.message);
                    }
                }
            });
        });
    }
}

module.exports = WebSocketService;
```

src\interfaces\controllers\eventcontroller.js

to top

```
const Event = require('../../domain/models/event.model');
const WorkoutSession = require('../../domain/models/workout-session.model');
const asyncHandler = require('../../application/middlewares/async.middleware');

// @desc    Get all events for the current user
// @route   GET /api/events
// @access  Private
const getEvents = asyncHandler(async (req, res) => {
  const events = await Event.find({
    $or: [
      { creator: req.user._id },
      { 'participants.user': req.user._id }
    ]
  }).populate('creator', 'name email').populate('participants.user', 'name email');

  res.status(200).json({
    success: true,
    count: events.length,
    data: events
  });
});

// @desc    Get events within a specific date range
// @route   GET /api/events/range?start=yyyy-mm-dd&end=yyyy-mm-dd
// @access  Private
const getEventsByDateRange = asyncHandler(async (req, res) => {
  const { start, end } = req.query;

  if (!start || !end) {
    return res.status(400).json({
      success: false,
      error: 'Please provide start and end dates'
    });
  }

  const startDate = new Date(start);
  const endDate = new Date(end);

  const events = await Event.find({
    $or: [
      { creator: req.user._id },
      { 'participants.user': req.user._id }
    ],
    start: { $gte: startDate },
    end: { $lte: endDate }
  }).populate('creator', 'name email').populate('participants.user', 'name email');

  res.status(200).json({
    success: true,
    count: events.length,
    data: events
  });
```

```
});

// @desc    Get single event by ID
// @route   GET /api/events/:id
// @access  Private
const getEventById = asyncHandler(async (req, res) => {
  const event = await Event.findById(req.params.id)
    .populate('creator', 'name email')
    .populate('participants.user', 'name email');

  if (!event) {
    return res.status(404).json({
      success: false,
      error: 'Event not found'
    });
  }

  // Check if user is creator or participant
  const isCreator = event.creator._id.toString() === req.user._id.toString();
  const isParticipant = event.participants.some(p =>
    p.user._id.toString() === req.user._id.toString()
  );

  if (!isCreator && !isParticipant) {
    return res.status(403).json({
      success: false,
      error: 'Not authorized to access this event'
    });
  }

  // If this is a workout session, populate the session details
  if (event.isWorkoutSession && event.sessionDetails) {
    const sessionDetails = await WorkoutSession.findById(event.sessionDetails);

    return res.status(200).json({
      success: true,
      data: {
        ...event._doc,
        sessionDetails
      }
    });
  }

  res.status(200).json({
    success: true,
    data: event
  });
});

// @desc    Create new event
// @route   POST /api/events
// @access  Private
const createEvent = asyncHandler(async (req, res) => {
```

```
  // Add user to request body
  req.body.creator = req.user._id;

  const { isWorkoutSession, sessionDetails, ...eventData } = req.body;

  // Create the event
  const event = await Event.create(eventData);

  // If this is a workout session, create the session details
  if (isWorkoutSession && sessionDetails) {
    // Add event ID to session details
    sessionDetails.eventId = event._id;

    const workoutSession = await WorkoutSession.create(sessionDetails);

    // Update event with session details reference
    event.isWorkoutSession = true;
    event.sessionDetails = workoutSession._id;
    await event.save();

    return res.status(201).json({
      success: true,
      data: {
        ...event._doc,
        sessionDetails: workoutSession
      }
    });
  }

  res.status(201).json({
    success: true,
    data: event
  });
});

// @desc    Update event
// @route   PUT /api/events/:id
// @access  Private
const updateEvent = asyncHandler(async (req, res) => {
  let event = await Event.findById(req.params.id);

  if (!event) {
    return res.status(404).json({
      success: false,
      error: 'Event not found'
    });
  }

  // Make sure user is event creator
  if (event.creator.toString() !== req.user._id.toString()) {
    return res.status(403).json({
      success: false,
      error: 'Not authorized to update this event'
```

```
    });
  }

  const { sessionDetails, ...eventData } = req.body;

  // Update event
  event = await Event.findByIdAndUpdate(req.params.id, eventData, {
    new: true,
    runValidators: true
  });

  // If this is a workout session, update session details
  if (event.isWorkoutSession && event.sessionDetails && sessionDetails) {
    await WorkoutSession.findByIdAndUpdate(event.sessionDetails, sessionDetails, {
      new: true,
      runValidators: true
    });
  }

  res.status(200).json({
    success: true,
    data: event
  });
});

// @desc    Delete event
// @route   DELETE /api/events/:id
// @access  Private
const deleteEvent = asyncHandler(async (req, res) => {
  const event = await Event.findById(req.params.id);

  if (!event) {
    return res.status(404).json({
      success: false,
      error: 'Event not found'
    });
  }

  // Make sure user is event creator
  if (event.creator.toString() !== req.user._id.toString()) {
    return res.status(403).json({
      success: false,
      error: 'Not authorized to delete this event'
    });
  }

  // If this is a workout session, delete session details
  if (event.isWorkoutSession && event.sessionDetails) {
    await WorkoutSession.findByIdAndDelete(event.sessionDetails);
  }

  await event.deleteOne();
```

```
  res.status(200).json({
    success: true,
    data: {}
  });
});

// @desc    Add participant to event
// @route   POST /api/events/:id/participants
// @access  Private
const addParticipant = asyncHandler(async (req, res) => {
  const { userId, status = 'pending' } = req.body;

  if (!userId) {
    return res.status(400).json({
      success: false,
      error: 'Please provide a user ID'
    });
  }

  const event = await Event.findById(req.params.id);

  if (!event) {
    return res.status(404).json({
      success: false,
      error: 'Event not found'
    });
  }

  // Check if user is already a participant
  const existingParticipant = event.participants.find(
    p => p.user.toString() === userId
  );

  if (existingParticipant) {
    return res.status(400).json({
      success: false,
      error: 'User is already a participant'
    });
  }

  // Add new participant
  event.participants.push({
    user: userId,
    status
  });

  await event.save();

  res.status(200).json({
    success: true,
    data: event
  });
});
```

```
// @desc    Update participant status
// @route   PUT /api/events/:id/participants/:userId
// @access  Private
const updateParticipantStatus = asyncHandler(async (req, res) => {
  const { status } = req.body;

  if (!status || !['pending', 'accepted', 'declined'].includes(status)) {
    return res.status(400).json({
      success: false,
      error: 'Please provide a valid status'
    });
  }

  const event = await Event.findById(req.params.id);

  if (!event) {
    return res.status(404).json({
      success: false,
      error: 'Event not found'
    });
  }

  // Find participant
  const participantIndex = event.participants.findIndex(
    p => p.user.toString() === req.params.userId
  );

  if (participantIndex === -1) {
    return res.status(404).json({
      success: false,
      error: 'Participant not found'
    });
  }

  // Update participant status
  event.participants[participantIndex].status = status;

  await event.save();

  res.status(200).json({
    success: true,
    data: event
  });
});

// @desc    Remove participant from event
// @route   DELETE /api/events/:id/participants/:userId
// @access  Private
const removeParticipant = asyncHandler(async (req, res) => {
  const event = await Event.findById(req.params.id);

  if (!event) {
```

```
      return res.status(404).json({
        success: false,
        error: 'Event not found'
      });
    }

    // Make sure user is event creator or removing themself
    const isCreator = event.creator.toString() === req.user._id.toString();
    const isSelfRemoval = req.params.userId === req.user._id.toString();

    if (!isCreator && !isSelfRemoval) {
      return res.status(403).json({
        success: false,
        error: 'Not authorized to remove this participant'
      });
    }

    // Remove participant
    event.participants = event.participants.filter(
      p => p.user.toString() !== req.params.userId
    );

    await event.save();

    res.status(200).json({
      success: true,
      data: event
    });
  });

  module.exports = {
    getEvents,
    getEventsByDateRange,
    getEventById,
    createEvent,
    updateEvent,
    deleteEvent,
    addParticipant,
    updateParticipantStatus,
    removeParticipant
  };
```

src\interfaces\controllers\messagecontroller.js

to top

```
const Message = require('../../domain/models/message.model');
const Event = require('../../domain/models/event.model');
const asyncHandler = require('../../application/middlewares/async.middleware');

// @desc    Get messages for a specific event
// @route   GET /api/messages/event/:eventId
// @access  Private
const getEventMessages = asyncHandler(async (req, res) => {
  const { eventId } = req.params;
  const { limit = 50, before } = req.query;

  // Verify the event exists and user has access
  const event = await Event.findById(eventId);
  if (!event) {
    return res.status(404).json({
      success: false,
      error: 'Event not found'
    });
  }

  // Check if user is creator or participant
  const isCreator = event.creator.toString() === req.user._id.toString();
  const isParticipant = event.participants.some(p =>
    p.user.toString() === req.user._id.toString() && p.status === 'accepted'
  );

  if (!isCreator && !isParticipant) {
    return res.status(403).json({
      success: false,
      error: 'Not authorized to access messages for this event'
    });
  }

  // Build query
  let query = { eventId };
  if (before) {
    query.createdAt = { $lt: new Date(before) };
  }

  // Get messages
  const messages = await Message.find(query)
    .sort({ createdAt: -1 })
    .limit(parseInt(limit))
    .populate('sender', 'name email')
    .lean();

  // Return messages in chronological order
  res.status(200).json({
    success: true,
    count: messages.length,
    data: messages.reverse()
  });
```

```
});

// @desc    Get recent messages across all events for the user
// @route   GET /api/messages/recent
// @access  Private
const getRecentMessages = asyncHandler(async (req, res) => {
  const { limit = 20 } = req.query;

  // Find events where user is creator or participant
  const events = await Event.find({
    $or: [
      { creator: req.user._id },
      { 'participants.user': req.user._id, 'participants.status': 'accepted' }
    ]
  }).select('_id');

  const eventIds = events.map(event => event._id);

  // Get recent messages from these events
  const messages = await Message.find({ eventId: { $in: eventIds } })
    .sort({ createdAt: -1 })
    .limit(parseInt(limit))
    .populate('sender', 'name email')
    .populate('eventId', 'title')
    .lean();

  res.status(200).json({
    success: true,
    count: messages.length,
    data: messages
  });
});

// @desc    Send a new message
// @route   POST /api/messages
// @access  Private
const sendMessage = asyncHandler(async (req, res) => {
  const { eventId, content } = req.body;

  if (!eventId || !content) {
    return res.status(400).json({
      success: false,
      error: 'Please provide event ID and message content'
    });
  }

  // Verify the event exists and user has access
  const event = await Event.findById(eventId);
  if (!event) {
    return res.status(404).json({
      success: false,
      error: 'Event not found'
    });
```

```
  }

  // Check if user is creator or participant
  const isCreator = event.creator.toString() === req.user._id.toString();
  const isParticipant = event.participants.some(p =>
    p.user.toString() === req.user._id.toString() && p.status === 'accepted'
  );

  if (!isCreator && !isParticipant) {
    return res.status(403).json({
      success: false,
      error: 'Not authorized to send messages to this event'
    });
  }

  // Create message
  const message = await Message.create({
    sender: req.user._id,
    content,
    eventId,
    readBy: [{ user: req.user._id }] // Mark as read by sender
  });

  // Populate sender info for immediate use in socket broadcast
  const populatedMessage = await Message.findById(message._id)
    .populate('sender', 'name email')
    .lean();

  // Return the populated message
  res.status(201).json({
    success: true,
    data: populatedMessage
  });

  // Notify Socket.IO of new message (handled by WebSocket service)
  if (req.app.get('io')) {
    req.app.get('io').to(`event:${eventId}`).emit('newMessage', populatedMessage);
  }
});

// @desc    Mark messages as read for a specific event
// @route   PUT /api/messages/read/event/:eventId
// @access  Private
const markMessagesAsRead = asyncHandler(async (req, res) => {
  const { eventId } = req.params;

  // Verify the event exists and user has access
  const event = await Event.findById(eventId);
  if (!event) {
    return res.status(404).json({
      success: false,
      error: 'Event not found'
    });
```

```
  }

  // Check if user is creator or participant
  const isCreator = event.creator.toString() === req.user._id.toString();
  const isParticipant = event.participants.some(p =>
    p.user.toString() === req.user._id.toString() && p.status === 'accepted'
  );

  if (!isCreator && !isParticipant) {
    return res.status(403).json({
      success: false,
      error: 'Not authorized to access messages for this event'
    });
  }

  // Find all unread messages for this event
  const result = await Message.updateMany(
    {
      eventId,
      sender: { $ne: req.user._id }, // Skip messages sent by the current user
      'readBy.user': { $ne: req.user._id } // Not already read by this user
    },
    {
      $push: { readBy: { user: req.user._id, readAt: new Date() } }
    }
  );

  res.status(200).json({
    success: true,
    data: {
      messagesMarkedAsRead: result.modifiedCount
    }
  });
});

module.exports = {
  getEventMessages,
  getRecentMessages,
  sendMessage,
  markMessagesAsRead
};
```

src\interfaces\http\routes\authRoutesjs

to top

```
const express = require('express');

function createAuthRoutes(authService, authProvider) {
    const router = express.Router();

    // Manual signup
    router.post('/signup', async (req, res) => {
        try {
            const { email, password, name } = req.body;
            const user = await authService.registerUser(email, password, name);

            req.login(user, (err) => {
                if (err) {
                    return res.status(500).json({ error: 'Error logging in after
signup' });
                }
                res.json({ message: 'Signup successful', user: user.toJSON() });
            });
        } catch (error) {
            res.status(400).json({ error: error.message });
        }
    });

    // Manual login
    router.post('/login', (req, res, next) => {
        authProvider.authenticate('local', (err, user, info) => {
            if (err) {
                return res.status(500).json({ error: 'Internal server error' });
            }
            if (!user) {
                return res.status(401).json({ error: 'Invalid email or password' });
            }
            req.login(user, (err) => {
                if (err) {
                    return res.status(500).json({ error: 'Error logging in' });
                }
                return res.json({ message: 'Login successful', user: user.toJSON() });
            });
        })(req, res, next);
    });

    // Google OAuth routes
    router.get('/google', authProvider.authenticate('google', {
        scope: ['profile', 'email']
    }));

    router.get('/google/callback',
        authProvider.authenticate('google', { failureRedirect: '/login' }),
        (req, res) => {
            res.redirect('/dashboard');
        }
    );
```

```
    // Logout
    router.get('/logout', (req, res) => {
        req.logout((err) => {
            if (err) {
                return res.status(500).json({ error: 'Error logging out' });
            }
            res.json({ message: 'Logged out successfully' });
        });
    });

    // Get current user
    router.get('/current-user', (req, res) => {
        if (!req.user) {
            return res.status(401).json({ error: 'Not authenticated' });
        }
        res.json({ user: req.user.toJSON() });
    });

    router.get('/check', (req, res) => {
        if (req.isAuthenticated()) {
            res.status(200).json({ authenticated: true });
        } else {
            res.status(401).json({ authenticated: false });
        }
    });

    return router;
}

module.exports = createAuthRoutes;
```

src\interfaces\http\routes\meetingRoutesjs

to top

```
const express = require('express');

function createMeetingRoutes(meetingService) {
    const router = express.Router();

    router.post('/meetings', async (req, res) => {
        try {
            const meeting = await meetingService.createMeeting();
            res.json({ meetingId: meeting.id });
        } catch (error) {
            res.status(500).json({ error: error.message });
        }
    });

    router.get('/meetings/:id', async (req, res) => {
        try {
            const meeting = await meetingService.getMeeting(req.params.id);
            if (!meeting) {
                return res.status(404).json({ error: 'Meeting not found' });
            }
            res.json(meeting);
        } catch (error) {
            res.status(500).json({ error: error.message });
        }
    });

    return router;
}

module.exports = createMeetingRoutes;
```

src\interfaces\routes\eventroutes.js

to top

```
const express = require('express');
const router = express.Router();
const eventController = require('../controllers/event.controller');
const { isAuthenticated } = require('../../application/middlewares/auth.middleware');

// Apply authentication middleware to all event routes
router.use(isAuthenticated);

// Get all events
router.get('/', eventController.getEvents);

// Get events in date range
router.get('/range', eventController.getEventsByDateRange);

// Get single event by ID
router.get('/:id', eventController.getEventById);

// Create new event
router.post('/', eventController.createEvent);

// Update event
router.put('/:id', eventController.updateEvent);

// Delete event
router.delete('/:id', eventController.deleteEvent);

// Manage event participants
router.post('/:id/participants', eventController.addParticipant);
router.put('/:id/participants/:userId', eventController.updateParticipantStatus);
router.delete('/:id/participants/:userId', eventController.removeParticipant);

module.exports = router;
```

src\interfaces\routes\messageroutes.js

to top

```
const express = require('express');
const router = express.Router();
const messageController = require('../controllers/message.controller');
const { isAuthenticated } = require('../../application/middlewares/auth.middleware');

// Apply authentication middleware to all message routes
router.use(isAuthenticated);

// Get messages for a specific event
router.get('/event/:eventId', messageController.getEventMessages);

// Get recent messages
router.get('/recent', messageController.getRecentMessages);

// Send a new message
router.post('/', messageController.sendMessage);

// Mark messages as read
router.put('/read/event/:eventId', messageController.markMessagesAsRead);

module.exports = router;
```

src\interfaces\web\appjs

to top

```
// Services are now loaded directly in HTML
// import WebRTCService from '/static/services/webrtc.service.js';
// import MeetingService from '/static/services/meeting.service.js';
// import calendarService from '/static/services/calendar.service.js';
// import messageService from '/static/services/message.service.js';

class App {
    constructor() {
        this.socket = io('/');
        this.webRTCService = new WebRTCService(this.socket);
        this.meetingService = new MeetingService();
        this.calendarService = calendarService;
        this.messageService = messageService;
        this.setupSocketHandlers();
        this.handleUrlMeeting();
        this.handleUrlErrors();
        this.currentEventId = null;
        this.setupChatHandler();
    }

    setupSocketHandlers() {
        this.socket.on('user-connected', async (userId) => {
            await this.webRTCService.connectToNewUser(userId);
        });

        this.socket.on('user-disconnected', (userId) => {
            this.webRTCService.closePeerConnection(userId);
        });

        this.socket.on('offer', async (offer, senderId) => {
            await this.webRTCService.handleOffer(offer, senderId);
        });

        this.socket.on('answer', async (answer, senderId) => {
            await this.webRTCService.handleAnswer(answer, senderId);
        });

        this.socket.on('ice-candidate', async (candidate, senderId) => {
            await this.webRTCService.handleIceCandidate(candidate, senderId);
        });
    }

    setupChatHandler() {
        // Add chat UI elements if not already present
        if (!document.getElementById('chatContainer')) {
            const videoGrid = document.getElementById('videoGrid');
            if (videoGrid) {
                // Create and append chat container after video grid
                const chatContainer = document.createElement('div');
                chatContainer.id = 'chatContainer';
                chatContainer.className = 'mt-6 bg-white rounded-lg shadow p-4';
                chatContainer.innerHTML = `
```

```
                    <h3 class="text-lg font-medium mb-2">Chat</h3>
                    <div id="chatMessages" class="h-64 overflow-y-auto mb-4 p-2 border
rounded-lg"></div>
                    <div class="flex">
                        <input
                            type="text"
                            id="chatInput"
                            placeholder="Type a message..."
                            class="flex-grow rounded-l-md border-gray-300 shadow-sm"
                        />
                        <button
                            id="sendMessageBtn"
                            class="bg-blue-600 text-white py-2 px-4 rounded-r-md
hover:bg-blue-700"
                        >
                            Send
                        </button>
                    </div>
                `;
                videoGrid.parentNode.insertBefore(chatContainer,
videoGrid.nextSibling);

                // Add event listener for sending messages
                document.getElementById('sendMessageBtn').addEventListener('click', ()
=> this.sendChatMessage());
                document.getElementById('chatInput').addEventListener('keypress', (e)
=> {
                    if (e.key === 'Enter') this.sendChatMessage();
                });
            }
        }

        // Register message listener
        this.messageService.registerMessageListener((message) => {
            this.displayChatMessage(message);
        });
    }

    async sendChatMessage() {
        const input = document.getElementById('chatInput');
        const content = input.value.trim();

        if (content && this.currentEventId) {
            await this.messageService.sendMessage(content);
            input.value = '';
        }
    }

    displayChatMessage(message) {
        const messagesContainer = document.getElementById('chatMessages');
        if (!messagesContainer) return;

        const messageElement = document.createElement('div');
```

```
        messageElement.className = 'mb-2';

        if (message.isSystem) {
            // System message (user joined/left)
            messageElement.innerHTML = `
                <div class="text-xs text-center text-gray-500 my-2">${message.content}
</div>
            `;
        } else {
            // Regular user message
            const isSelf = message.sender?._id === this.getCurrentUserId();
            messageElement.innerHTML = `
                <div class="${isSelf ? 'text-right' : ''}">
                    <div class="inline-block rounded-lg py-2 px-3 ${isSelf ? 'bg-blue-
100' : 'bg-gray-100'}">
                        ${isSelf ? '' : `<div class="text-xs text-gray-600 font-
semibold">${message.sender?.name || 'User'}</div>`}
                        <div>${message.content}</div>
                    </div>
                    <div class="text-xs text-gray-500 mt-1">
                        ${new Date(message.timestamp).toLocaleTimeString([], {hour: '2-
digit', minute:'2-digit'})}
                    </div>
                </div>
            `;
        }

        messagesContainer.appendChild(messageElement);
        messagesContainer.scrollTop = messagesContainer.scrollHeight;
    }

    async createMeeting() {
        try {
            const meetingId = await this.meetingService.createMeeting();
            await this.setupMediaAndJoinMeeting(meetingId);
            window.history.pushState({}, '', `/meeting/${meetingId}`);
            this.showMessage(`Meeting created! Share this URL to invite others`,
'success');
        } catch (error) {
            this.showMessage(error.message, 'error');
        }
    }

    handleUrlMeeting() {
        const path = window.location.pathname;
        const meetingMatch = path.match(/^\/meeting\/([^\/]+)/);

        if (meetingMatch) {
            const meetingId = meetingMatch[1];
            this.setupMediaAndJoinMeeting(meetingId).catch(error => {
                this.showMessage(error.message, 'error');
            });
        }
```

```
    }

    handleUrlErrors() {
        const urlParams = new URLSearchParams(window.location.search);
        const error = urlParams.get('error');

        if (error === 'meeting-not-found') {
            this.showMessage('The meeting you tried to join does not exist.', 'error');
        } else if (error === 'invalid-meeting') {
            this.showMessage('Unable to join the meeting. Please try again or create a
new meeting.', 'error');
        }
    }

    showJoinSection() {
        document.getElementById('joinSection').style.display = 'block';
    }

    async joinMeeting() {
        const meetingCode = document.getElementById('meetingCode').value.trim();
        if (!meetingCode) {
            this.showMessage('Please enter a meeting code', 'error');
            return;
        }

        try {
            await this.meetingService.joinMeeting(meetingCode);
            await this.setupMediaAndJoinMeeting(meetingCode);
            this.showMessage(`Successfully joined meeting ${meetingCode}`, 'success');
        } catch (error) {
            this.showMessage(error.message, 'error');
        }
    }

    async joinWorkoutSession(eventId) {
        if (!eventId) return;

        try {
            // Get event details
            const response = await fetch(`/api/events/${eventId}`);
            if (!response.ok) throw new Error('Failed to load event details');

            const event = await response.json();

            if (!event.workoutSession || !event.workoutSession.videoCallEnabled) {
                this.showMessage('Video calls are not enabled for this session',
'error');
                return;
            }

            // Join the event chat
            this.currentEventId = eventId;
            this.messageService.joinEventChat(eventId);
```

```
            // Setup media for video call
            await this.setupMediaAndJoinMeeting(eventId);

            // Change heading to show event title
            const sessionTitle = document.createElement('h3');
            sessionTitle.textContent = `${event.title}`;
            sessionTitle.className = 'text-lg font-medium mb-4';

            const videoGrid = document.getElementById('videoGrid');
            if (videoGrid && videoGrid.parentNode) {
                // Insert before video grid
                videoGrid.parentNode.insertBefore(sessionTitle, videoGrid);
            }

            this.showMessage(`Joined workout session: ${event.title}`, 'success');

            // Switch to home tab to show the video
            document.getElementById('navHome').click();

        } catch (error) {
            console.error('Error joining workout session:', error);
            this.showMessage('Failed to join workout session: ' + error.message,
'error');
        }
    }

    async setupMediaAndJoinMeeting(meetingId) {
        try {
            const stream = await this.webRTCService.setupMediaStream();
            const videoElement = this.webRTCService.createVideoElement(stream, true);
            document.getElementById('videoGrid').appendChild(videoElement);
            this.socket.emit('join-room', meetingId);
            this.showMeetingControls();
            await this.populateCameraList();
        } catch (error) {
            this.showMessage(error.message, 'error');
        }
    }

    async populateCameraList() {
        const cameras = await this.webRTCService.getAvailableCameras();
        const cameraSelect = document.getElementById('cameraSelect');
        cameraSelect.innerHTML = '<option value="">Select Camera</option>';

        cameras.forEach(camera => {
            const option = document.createElement('option');
            option.value = camera.deviceId;
            option.text = camera.label || `Camera ${cameraSelect.length}`;
            cameraSelect.appendChild(option);
        });

        if (cameras.length > 1) {
```

```
            cameraSelect.style.display = 'inline-block';
        }
    }

    async switchCamera(deviceId) {
        if (!deviceId) return;

        try {
            await this.webRTCService.switchCamera(deviceId);
            this.showMessage('Camera switched successfully', 'success');
        } catch (error) {
            this.showMessage('Failed to switch camera: ' + error.message, 'error');
        }
    }

    showMeetingControls() {
        document.getElementById('initialButtons').style.display = 'none';
        document.getElementById('joinSection').style.display = 'none';
        document.getElementById('sessionControls').style.display = 'block';
    }

    async toggleVideo() {
        const isEnabled = await this.webRTCService.toggleVideo();
        document.getElementById('videoBtn').textContent = isEnabled ? 'Turn Off Video'
: 'Turn On Video';
    }

    toggleAudio() {
        const isEnabled = this.webRTCService.toggleAudio();
        document.getElementById('audioBtn').textContent = isEnabled ? 'Mute Audio' :
'Unmute Audio';
    }

    leaveMeeting() {
        // Clean up WebRTC connections and media
        this.webRTCService.cleanup();

        // Leave event chat if in one
        if (this.currentEventId) {
            this.messageService.leaveEventChat();
            this.currentEventId = null;
        }

        // Disconnect socket
        this.socket.disconnect();

        // Clear video grid
        const videoGrid = document.getElementById('videoGrid');
        while (videoGrid.firstChild) {
            videoGrid.removeChild(videoGrid.firstChild);
        }

        // Remove any session title
```

```
        const sessionTitle = videoGrid.previousElementSibling;
        if (sessionTitle && sessionTitle.tagName === 'H3') {
            sessionTitle.remove();
        }

        // Clear chat messages
        const chatMessages = document.getElementById('chatMessages');
        if (chatMessages) {
            chatMessages.innerHTML = '';
        }

        // Hide meeting controls and show initial buttons
        document.getElementById('sessionControls').style.display = 'none';
        document.getElementById('initialButtons').style.display = 'block';
    }

    showMessage(text, type) {
        const messageDiv = document.getElementById('message');
        messageDiv.textContent = text;
        messageDiv.className = `message ${type}`;
        messageDiv.style.display = 'block';

        // Hide the message after 5 seconds
        setTimeout(() => {
            messageDiv.style.display = 'none';
        }, 5000);
    }

    getCurrentUserId() {
        // This function should return the current user's ID
        // We'll try to get it from the profile picture URL or username element
        const userNameEl = document.getElementById('userName');
        if (userNameEl && userNameEl.dataset && userNameEl.dataset.userId) {
            return userNameEl.dataset.userId;
        }

        // Return null if we can't determine the user ID
        return null;
    }
}

// Initialize the app when the DOM is loaded
document.addEventListener('DOMContentLoaded', () => {
    window.app = new App();
});

// Add route handler for direct meeting access
if (typeof express !== 'undefined') {
    const app = express();

    app.get('/meeting/:id', (req, res) => {
        res.sendFile(path.join(__dirname, 'dashboard', 'pages', 'index.html'));
```

```
    });
}
```

src\interfaces\web\auth\pages\loginhtml

to top

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Login - Workoutmate</title>
    <link
      href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css"
      rel="stylesheet"
    />
    <link href="/static/auth/styles/auth.css" rel="stylesheet" />
  </head>
  <body class="bg-gray-100 min-h-screen flex items-center justify-center">
    <div class="auth-container">
      <h1 class="auth-title">Login to Workoutmate</h1>

      <!-- Manual Login Form -->
      <form id="loginForm" class="auth-form">
        <div class="auth-input-group">
          <label for="email" class="auth-input-label">Email</label>
          <input
            type="email"
            id="email"
            name="email"
            required
            class="auth-input"
          />
        </div>

        <div class="auth-input-group">
          <label for="password" class="auth-input-label">Password</label>
          <input
            type="password"
            id="password"
            name="password"
            required
            class="auth-input"
          />
        </div>

        <button type="submit" class="auth-button">Login</button>
      </form>

      <div class="auth-divider">
        <span class="auth-divider-text">or</span>
      </div>

      <!-- Google Login Button -->
      <a href="/auth/google" class="auth-social-button">
        <img
          src="https://www.google.com/favicon.ico"
          alt="Google"
```

```
          class="w-5 h-5"
        />
        Continue with Google
      </a>

      <p class="auth-footer">
        Don't have an account?
        <a href="/signup" class="auth-link">Sign up</a>
      </p>

      <div id="message" class="message" style="display: none"></div>
    </div>

    <script type="module" src="/static/auth/scripts/auth.js"></script>
  </body>
</html>
```

src\interfaces\web\auth\pages\signuphtml

to top

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Sign Up - Workoutmate</title>
    <link
      href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css"
      rel="stylesheet"
    />
    <link href="/static/auth/styles/auth.css" rel="stylesheet" />
  </head>
  <body class="bg-gray-100 min-h-screen flex items-center justify-center">
    <div class="auth-container">
      <h1 class="auth-title">Create an Account</h1>

      <!-- Manual Signup Form -->
      <form id="signupForm" class="auth-form">
        <div class="auth-input-group">
          <label for="name" class="auth-input-label">Full Name</label>
          <input
            type="text"
            id="name"
            name="name"
            required
            class="auth-input"
          />
        </div>

        <div class="auth-input-group">
          <label for="email" class="auth-input-label">Email</label>
          <input
            type="email"
            id="email"
            name="email"
            required
            class="auth-input"
          />
        </div>

        <div class="auth-input-group">
          <label for="password" class="auth-input-label">Password</label>
          <input
            type="password"
            id="password"
            name="password"
            required
            class="auth-input"
          />
        </div>

        <div class="auth-input-group">
```

```
          <label for="confirmPassword" class="auth-input-label"
            >Confirm Password</label
          >
          <input
            type="password"
            id="confirmPassword"
            name="confirmPassword"
            required
            class="auth-input"
          />
        </div>

        <button type="submit" class="auth-button">Sign Up</button>
      </form>

      <div class="auth-divider">
        <span class="auth-divider-text">or</span>
      </div>

      <!-- Google Signup Button -->
      <a href="/auth/google" class="auth-social-button">
        <img
          src="https://www.google.com/favicon.ico"
          alt="Google"
          class="w-5 h-5"
        />
        Continue with Google
      </a>

      <p class="auth-footer">
        Already have an account?
        <a href="/login" class="auth-link">Log in</a>
      </p>

      <div id="message" class="message" style="display: none"></div>
    </div>

    <script type="module" src="/static/auth/scripts/auth.js"></script>
  </body>
</html>
```

src\interfaces\web\auth\scripts\authjs

to top

```
class AuthService {
    constructor() {
        this.setupEventListeners();
    }

    setupEventListeners() {
        const loginForm = document.getElementById('loginForm');
        const signupForm = document.getElementById('signupForm');

        if (loginForm) {
            loginForm.addEventListener('submit', (e) => this.handleLogin(e));
        }

        if (signupForm) {
            signupForm.addEventListener('submit', (e) => this.handleSignup(e));
        }
    }

    async handleLogin(event) {
        event.preventDefault();
        const email = document.getElementById('email').value;
        const password = document.getElementById('password').value;

        try {
            const response = await fetch('/auth/login', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ email, password })
            });

            const data = await response.json();

            if (response.ok) {
                window.location.href = '/dashboard';
            } else {
                this.showMessage(data.error || 'Login failed', 'error');
            }
        } catch (error) {
            this.showMessage('An error occurred during login', 'error');
            console.error('Login error:', error);
        }
    }

    async handleSignup(event) {
        event.preventDefault();
        const name = document.getElementById('name').value;
        const email = document.getElementById('email').value;
        const password = document.getElementById('password').value;
        const confirmPassword = document.getElementById('confirmPassword').value;
```

```
        if (password !== confirmPassword) {
            this.showMessage('Passwords do not match', 'error');
            return;
        }

        try {
            const response = await fetch('/auth/signup', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ name, email, password })
            });

            const data = await response.json();

            if (response.ok) {
                window.location.href = '/dashboard';
            } else {
                this.showMessage(data.error || 'Signup failed', 'error');
            }
        } catch (error) {
            this.showMessage('An error occurred during signup', 'error');
            console.error('Signup error:', error);
        }
    }

    showMessage(text, type) {
        const messageDiv = document.getElementById('message');
        if (messageDiv) {
            messageDiv.textContent = text;
            messageDiv.className = `message ${type}`;
            messageDiv.style.display = 'block';

            // Hide message after 5 seconds
            setTimeout(() => {
                messageDiv.style.display = 'none';
            }, 5000);
        }
    }
}

// Initialize auth service when the DOM is loaded
document.addEventListener('DOMContentLoaded', () => {
    new AuthService();
});
```

src\interfaces\web\dashboard\pages\indexhtml

to top

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Workoutmate</title>
    <link rel="stylesheet" href="/static/styles/main.css" />
    <link
      href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css"
      rel="stylesheet"
    />
    <!-- FullCalendar CSS -->
    <link href="https://cdn.jsdelivr.net/npm/@fullcalendar/core/main.min.css"
rel="stylesheet" />
    <link href="https://cdn.jsdelivr.net/npm/@fullcalendar/daygrid/main.min.css"
rel="stylesheet" />
    <link href="https://cdn.jsdelivr.net/npm/@fullcalendar/timegrid/main.min.css"
rel="stylesheet" />
  </head>
  <body class="bg-gray-100 min-h-screen">
    <nav class="bg-white shadow-sm">
      <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <div class="flex justify-between h-16">
          <div class="flex items-center">
            <h1 class="text-xl font-semibold">Workoutmate</h1>
            <!-- Navigation tabs -->
            <div class="ml-10 flex space-x-8">
              <a href="#" id="navHome" class="inline-flex items-center px-1 pt-1
border-b-2 border-blue-500 text-sm font-medium leading-5 text-gray-900 focus:outline-
none focus:border-blue-700 transition duration-150 ease-in-out">
                Home
              </a>
              <a href="#" id="navCalendar" class="inline-flex items-center px-1 pt-1
border-b-2 border-transparent text-sm font-medium leading-5 text-gray-500 hover:text-
gray-700 hover:border-gray-300 focus:outline-none focus:text-gray-700 focus:border-
gray-300 transition duration-150 ease-in-out">
                Calendar
              </a>
              <a href="#" id="navMeetings" class="inline-flex items-center px-1 pt-1
border-b-2 border-transparent text-sm font-medium leading-5 text-gray-500 hover:text-
gray-700 hover:border-gray-300 focus:outline-none focus:text-gray-700 focus:border-
gray-300 transition duration-150 ease-in-out">
                Meetings
              </a>
            </div>
          </div>
          <div class="flex items-center">
            <div class="flex items-center space-x-4">
              <img id="userAvatar" class="h-8 w-8 rounded-full" src="" alt="" />
              <span id="userName" class="text-gray-700"></span>
              <button id="logoutBtn" class="text-gray-700 hover:text-gray-900">
                Logout
```

```
              </button>
            </div>
          </div>
        </div>
      </div>
    </nav>

    <main class="max-w-7xl mx-auto py-6 sm:px-6 lg:px-8">
      <div class="px-4 py-6 sm:px-0">
        <!-- Home section -->
        <div id="homeSection" class="bg-white shadow rounded-lg p-6">
          <div class="container">
            <h2 class="text-lg font-medium mb-4">Welcome to your Dashboard</h2>
            <p class="text-gray-600 mb-6">
              You are now logged in and can access all features of Workoutmate.
            </p>
            <div id="initialButtons" class="buttons space-y-4">
              <button
                onclick="app.createMeeting()"
                class="w-full bg-blue-600 text-white py-2 px-4 rounded-md hover:bg-
blue-700"
              >
                Create Meeting
              </button>
              <button
                onclick="app.showJoinSection()"
                class="w-full bg-blue-600 text-white py-2 px-4 rounded-md hover:bg-
blue-700"
              >
                Join Meeting
              </button>
            </div>
            <div id="joinSection" class="space-y-4 mt-4" style="display: none">
              <input
                type="text"
                id="meetingCode"
                placeholder="Enter meeting code"
                class="w-full rounded-md border-gray-300 shadow-sm"
              />
              <button
                onclick="app.joinMeeting()"
                class="w-full bg-blue-600 text-white py-2 px-4 rounded-md hover:bg-
blue-700"
              >
                Join
              </button>
            </div>
            <div
              id="sessionControls"
              class="space-y-4 mt-4"
              style="display: none"
            >
              <button
```

```
                onclick="app.toggleVideo().catch(error =>
app.showMessage(error.message, 'error'))"
                id="videoBtn"
                class="w-full bg-blue-600 text-white py-2 px-4 rounded-md hover:bg-
blue-700"
              >
                Turn Off Video
              </button>
              <button
                onclick="app.toggleAudio()"
                id="audioBtn"
                class="w-full bg-blue-600 text-white py-2 px-4 rounded-md hover:bg-
blue-700"
              >
                Mute Audio
              </button>
              <select
                id="cameraSelect"
                onchange="app.switchCamera(this.value)"
                style="display: none"
                class="w-full rounded-md border-gray-300 shadow-sm"
              >
                <option value="">Select Camera</option>
              </select>
              <button
                onclick="app.leaveMeeting()"
                class="w-full bg-red-600 text-white py-2 px-4 rounded-md hover:bg-red-
700"
              >
                Leave Session
              </button>
            </div>
            <div id="videoGrid" class="mt-4"></div>
            <div id="message" class="message mt-4" style="display: none"></div>
          </div>
        </div>

        <!-- Calendar section -->
        <div id="calendarSection" class="bg-white shadow rounded-lg p-6"
style="display: none">
          <div class="container">
            <h2 class="text-lg font-medium mb-4">Your Workout Calendar</h2>
            <p class="text-gray-600 mb-6">
              Schedule and manage your workout sessions.
            </p>
            <div class="flex justify-end mb-4">
              <button
                id="createEventBtn"
                class="bg-blue-600 text-white py-2 px-4 rounded-md hover:bg-blue-700"
              >
                Create Event
              </button>
            </div>
```

```
            <div id="calendar"></div>
          </div>
        </div>

        <!-- Meetings section -->
        <div id="meetingsSection" class="bg-white shadow rounded-lg p-6"
style="display: none">
          <div class="container">
            <h2 class="text-lg font-medium mb-4">Your Workout Sessions</h2>
            <p class="text-gray-600 mb-6">
              View and join your scheduled workout sessions.
            </p>
            <div id="meetingsList" class="space-y-4">
              <!-- Meeting items will be added here dynamically -->
            </div>
          </div>
        </div>
      </div>
    </main>

    <!-- Event Modal -->
    <div id="eventModal" class="fixed inset-0 bg-gray-500 bg-opacity-75 flex items-
center justify-center hidden">
      <div class="bg-white rounded-lg p-6 max-w-lg w-full">
        <h3 class="text-lg font-medium mb-4" id="eventModalTitle">Create Event</h3>
        <form id="eventForm">
          <div class="space-y-4">
            <div>
              <label for="eventTitle" class="block text-sm font-medium text-gray-
700">Title</label>
              <input type="text" id="eventTitle" name="title" class="mt-1 block w-full
rounded-md border-gray-300 shadow-sm" required>
            </div>
            <div>
              <label for="eventDescription" class="block text-sm font-medium text-gray-
700">Description</label>
              <textarea id="eventDescription" name="description" rows="3" class="mt-1
block w-full rounded-md border-gray-300 shadow-sm"></textarea>
            </div>
            <div class="grid grid-cols-2 gap-4">
              <div>
                <label for="eventStart" class="block text-sm font-medium text-gray-
700">Start Date/Time</label>
                <input type="datetime-local" id="eventStart" name="start" class="mt-1
block w-full rounded-md border-gray-300 shadow-sm" required>
              </div>
              <div>
                <label for="eventEnd" class="block text-sm font-medium text-gray-
700">End Date/Time</label>
                <input type="datetime-local" id="eventEnd" name="end" class="mt-1 block
w-full rounded-md border-gray-300 shadow-sm" required>
              </div>
            </div>
```

```
            <div>
              <label class="block text-sm font-medium text-gray-700">Event Type</label>
              <div class="mt-1">
                <label class="inline-flex items-center">
                  <input type="checkbox" id="isWorkoutSession" name="isWorkoutSession"
class="rounded border-gray-300 text-blue-600 shadow-sm">
                  <span class="ml-2">This is a workout session</span>
                </label>
              </div>
            </div>
            <div id="workoutSessionFields" class="space-y-4 hidden">
              <div>
                <label for="workoutType" class="block text-sm font-medium text-gray-
700">Workout Type</label>
                <select id="workoutType" name="workoutType" class="mt-1 block w-full
rounded-md border-gray-300 shadow-sm">
                  <option value="cardio">Cardio</option>
                  <option value="strength">Strength</option>
                  <option value="flexibility">Flexibility</option>
                  <option value="hiit">HIIT</option>
                  <option value="yoga">Yoga</option>
                  <option value="other">Other</option>
                </select>
              </div>
              <div>
                <label for="intensity" class="block text-sm font-medium text-gray-
700">Intensity</label>
                <select id="intensity" name="intensity" class="mt-1 block w-full
rounded-md border-gray-300 shadow-sm">
                  <option value="low">Low</option>
                  <option value="medium">Medium</option>
                  <option value="high">High</option>
                </select>
              </div>
              <div>
                <label class="block text-sm font-medium text-gray-700">Video
Call</label>
                <div class="mt-1">
                  <label class="inline-flex items-center">
                    <input type="checkbox" id="videoCallEnabled"
name="videoCallEnabled" class="rounded border-gray-300 text-blue-600 shadow-sm"
checked>
                    <span class="ml-2">Enable video call for this session</span>
                  </label>
                </div>
              </div>
            </div>
          </div>
          <div class="mt-6 flex justify-end space-x-3">
            <button type="button" id="closeEventModal" class="bg-gray-100 text-gray-700
py-2 px-4 rounded-md hover:bg-gray-200">
              Cancel
            </button>
```

```
            <button type="submit" class="bg-blue-600 text-white py-2 px-4 rounded-md
hover:bg-blue-700">
              Save
            </button>
          </div>
        </form>
      </div>
    </div>

    <script src="/socket.io/socket.io.js"></script>

    <!-- FullCalendar JS -->
    <script src="https://cdn.jsdelivr.net/npm/@fullcalendar/core/main.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@fullcalendar/daygrid/main.min.js">
</script>
    <script src="https://cdn.jsdelivr.net/npm/@fullcalendar/timegrid/main.min.js">
</script>
    <script src="https://cdn.jsdelivr.net/npm/@fullcalendar/interaction/main.min.js">
</script>

    <!-- Load app.js as a regular script, not a module -->
    <script src="/static/services/webrtc.service.js"></script>
    <script src="/static/services/meeting.service.js"></script>
    <script src="/static/services/calendar.service.js"></script>
    <script src="/static/services/message.service.js"></script>
    <script src="/static/app.js"></script>

    <script>
      // Fetch user data when the page loads
      async function fetchUserData() {
        try {
          const response = await fetch("/auth/current-user");
          const data = await response.json();

          if (response.ok) {
            document.getElementById("userName").textContent = data.user.name;
            if (data.user.profilePicture) {
              document.getElementById("userAvatar").src =
                data.user.profilePicture;
            } else {
              document.getElementById("userAvatar").src =
                "https://www.gravatar.com/avatar/?d=mp";
            }
          } else {
            window.location.href = "/login";
          }
        } catch (error) {
          console.error("Error fetching user data:", error);
          window.location.href = "/login";
        }
      }

      // Handle logout
```

```
      document
        .getElementById("logoutBtn")
        .addEventListener("click", async () => {
          try {
            const response = await fetch("/auth/logout");
            if (response.ok) {
              window.location.href = "/login";
            }
          } catch (error) {
            console.error("Error logging out:", error);
          }
        });

      // Navigation handling
      document.getElementById("navHome").addEventListener("click", (e) => {
        e.preventDefault();
        showSection("homeSection");
        updateActiveNav("navHome");
      });

      document.getElementById("navCalendar").addEventListener("click", (e) => {
        e.preventDefault();
        showSection("calendarSection");
        updateActiveNav("navCalendar");
      });

      document.getElementById("navMeetings").addEventListener("click", (e) => {
        e.preventDefault();
        showSection("meetingsSection");
        updateActiveNav("navMeetings");
      });

      function showSection(sectionId) {
        // Hide all sections
        document.getElementById("homeSection").style.display = "none";
        document.getElementById("calendarSection").style.display = "none";
        document.getElementById("meetingsSection").style.display = "none";

        // Show the selected section
        document.getElementById(sectionId).style.display = "block";
      }

      function updateActiveNav(navId) {
        // Remove active class from all nav links
        document.getElementById("navHome").className = "inline-flex items-center px-1
pt-1 border-b-2 border-transparent text-sm font-medium leading-5 text-gray-500
hover:text-gray-700 hover:border-gray-300 focus:outline-none focus:text-gray-700
focus:border-gray-300 transition duration-150 ease-in-out";
        document.getElementById("navCalendar").className = "inline-flex items-center
px-1 pt-1 border-b-2 border-transparent text-sm font-medium leading-5 text-gray-500
hover:text-gray-700 hover:border-gray-300 focus:outline-none focus:text-gray-700
focus:border-gray-300 transition duration-150 ease-in-out";
        document.getElementById("navMeetings").className = "inline-flex items-center
```

```
px-1 pt-1 border-b-2 border-transparent text-sm font-medium leading-5 text-gray-500
hover:text-gray-700 hover:border-gray-300 focus:outline-none focus:text-gray-700
focus:border-gray-300 transition duration-150 ease-in-out";

        // Add active class to selected nav link
        document.getElementById(navId).className = "inline-flex items-center px-1 pt-1
border-b-2 border-blue-500 text-sm font-medium leading-5 text-gray-900 focus:outline-
none focus:border-blue-700 transition duration-150 ease-in-out";
      }

      // Load user data when the page loads
      fetchUserData();
    </script>
  </body>
</html>
```

src\interfaces\web\dashboard\scripts\dashboardjs

to top

```
import AuthUtils from '@/interfaces/web/shared/scripts/authUtils.js';

class DashboardPage {
    constructor() {
        this.init();
    }

    async init() {
        // Check authentication before initializing dashboard
        const isAuthenticated = await AuthUtils.checkAuthentication();
        if (!isAuthenticated) return;

        // Initialize dashboard components
        this.setupEventListeners();
        this.loadDashboardData();
    }

    setupEventListeners() {
        // Add your event listeners here
    }

    async loadDashboardData() {
        // Load dashboard data here
    }
}

// Initialize dashboard
new DashboardPage();
```

src\interfaces\web\react-app\eslintconfig.js

to top

```
import js from '@eslint/js'
import globals from 'globals'
import react from 'eslint-plugin-react'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'

export default [
  { ignores: ['dist'] },
  {
    files: ['**/*.{js,jsx}'],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
      parserOptions: {
        ecmaVersion: 'latest',
        ecmaFeatures: { jsx: true },
        sourceType: 'module',
      },
    },
    settings: { react: { version: '18.3' } },
    plugins: {
      react,
      'react-hooks': reactHooks,
      'react-refresh': reactRefresh,
    },
    rules: {
      ...js.configs.recommended.rules,
      ...react.configs.recommended.rules,
      ...react.configs['jsx-runtime'].rules,
      ...reactHooks.configs.recommended.rules,
      'react/jsx-no-target-blank': 'off',
      'react-refresh/only-export-components': [
        'warn',
        { allowConstantExport: true },
      ],
    },
  },
]
```

src\interfaces\web\react-app\indexhtml

to top

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Workoutmate</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

src\interfaces\web\react-app\public\vitesvg

to top

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
aria-hidden="true" role="img" class="iconify iconify--logos" width="31.88" height="32"
preserveAspectRatio="xMidYMid meet" viewBox="0 0 256 257"><defs><linearGradient
id="IconifyId1813088fe1fbc01fb466" x1="-.828%" x2="57.636%" y1="7.652%" y2="78.411%">
<stop offset="0%" stop-color="#41D1FF"></stop><stop offset="100%" stop-color="#BD34FE">
</stop></linearGradient><linearGradient id="IconifyId1813088fe1fbc01fb467" x1="43.376%"
x2="50.316%" y1="2.242%" y2="89.03%"><stop offset="0%" stop-color="#FFEA83"></stop>
<stop offset="8.333%" stop-color="#FFDD35"></stop><stop offset="100%" stop-
color="#FFA800"></stop></linearGradient></defs><path
fill="url(#IconifyId1813088fe1fbc01fb466)" d="M255.153 37.938L134.897 252.976c-2.483
4.44-8.862 4.466-11.382.048L.875 37.958c-2.746-4.814 1.371-10.646 6.827-9.67l120.385
21.517a6.537 6.537 0 0 0 2.322-.004l117.867-21.483c5.438-.991 9.574 4.796 6.877 9.62Z">
</path><path fill="url(#IconifyId1813088fe1fbc01fb467)" d="M185.432.063L96.44
17.501a3.268 3.268 0 0 0-2.634 3.014l-5.474 92.456a3.268 3.268 0 0 0 3.997
3.378l24.777-5.718c2.318-.535 4.413 1.507 3.936 3.838l-7.361 36.047c-.495 2.426 1.782
4.5 4.151 3.78l15.304-4.649c2.372-.72 4.652 1.36 4.15 3.788l-11.698 56.621c-.732 3.542
3.979 5.473 5.943 2.437l1.313-2.028l72.516-144.72c1.215-2.423-.88-5.186-3.54-4.672l-
25.505 4.922c-2.396.462-4.435-1.77-3.759-4.114l16.646-57.705c.677-2.35-1.37-4.583-
3.769-4.113Z"></path></svg>
```

src\interfaces\web\react-app\READMEmd

to top

React + Vite

This template provides a minimal setup to get React working in Vite with HMR and some ESLint rules.

Currently, two official plugins are available:

- @vitejs/plugin-react (https://github.com/vitejs/vite-plugin-react/blob/main/packages/plugin-react/README.md) uses Babel (https://babeljs.io/) for Fast Refresh
- @vitejs/plugin-react-swc (https://github.com/vitejs/vite-plugin-react-swc) uses SWC (https://swc.rs/) for Fast Refresh

src\interfaces\web\react-app\src\Appjsx

to top

```
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom'
import { Login } from './application/components/Login'
import { Home } from './application/components/Home'
import './App.css'

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/login" element={<Login />} />
        <Route path="/" element={<Home />} />
      </Routes>
    </Router>
  )
}

export default App
```

src\interfaces\web\react-app\src\application\components\Homejsx

to top

```jsx
import { useState } from 'react'
import { VideoCall } from './VideoCall'
import './VideoCall.css'
import reactLogo from '../../assets/react.svg'
import viteLogo from '/vite.svg'

const Home = () => {
  const [count, setCount] = useState(0)

  return (
    <div>
      <div>
        <a href="https://vite.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Workoutmate Video Call</h1>
      <VideoCall />
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.jsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </div>
  )
}

export { Home }
```

src\interfaces\web\react-app\src\application\components\Loginjsx

to top

```
const Login = () => {
  return (
    <div>
      <h1>Hello login!</h1>
    </div>
  )
}

export { Login }
```

src\interfaces\web\react-app\src\application\components\VideoCalljsx

to top

```jsx
import { useState, useRef, useEffect } from 'react';
import './VideoCall.css';

const VideoCall = () => {
  const [isCallActive, setIsCallActive] = useState(false);
  const [isBlurred, setIsBlurred] = useState(false);
  const videoRef = useRef(null);
  const streamRef = useRef(null);

  const startCall = async () => {
    try {
      const stream = await navigator.mediaDevices.getUserMedia({
        video: true,
        audio: true
      });
      streamRef.current = stream;
      if (videoRef.current) {
        videoRef.current.srcObject = stream;
      }
      setIsCallActive(true);
    } catch (error) {
      console.error('Error accessing camera:', error);
    }
  };

  const endCall = () => {
    if (streamRef.current) {
      streamRef.current.getTracks().forEach(track => track.stop());
    }
    if (videoRef.current) {
      videoRef.current.srcObject = null;
    }
    setIsCallActive(false);
    setIsBlurred(false);
  };

  const toggleBlur = () => {
    setIsBlurred(!isBlurred);
  };

  useEffect(() => {
    return () => {
      // Cleanup when component unmounts
      endCall();
    };
  }, []);

  return (
    <div className="video-call-container">
      <video
        ref={videoRef}
        autoPlay
```

```
        playsInline
        muted
        className={`video-feed ${isBlurred ? 'blurred' : ''}`}
      />
      <div className="controls">
        {!isCallActive ? (
          <button onClick={startCall} className="call-btn start-call">
            <span className="btn-icon">📞</span>
            Start Call
          </button>
        ) : (
          <div className="active-call-controls">
            <button
              onClick={toggleBlur}
              className={`call-btn blur-btn ${isBlurred ? 'active' : ''}`}
            >
              <span className="btn-icon">{isBlurred ? '👁' : '👁'}</span>
              {isBlurred ? 'Unblur Camera' : 'Blur Camera'}
            </button>
            <button onClick={endCall} className="call-btn end-call">
              <span className="btn-icon">❌</span>
              End Call
            </button>
          </div>
        )}
      </div>
    </div>
  );
};

export { VideoCall };
```

src\interfaces\web\react-app\src\assets\reactsvg

to top

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
aria-hidden="true" role="img" class="iconify iconify--logos" width="35.93" height="32"
preserveAspectRatio="xMidYMid meet" viewBox="0 0 256 228"><path fill="#00D8FF"
d="M210.483 73.824a171.49 171.49 0 0 0-8.24-2.597c.465-1.9.893-3.777 1.273-5.621c6.238-
30.281 2.16-54.676-11.769-62.708c-13.355-7.7-35.196.329-57.254 19.526a171.23 171.23 0 0
0-6.375 5.848a155.866 155.866 0 0 0-4.241-3.917C100.759 3.829 77.587-4.822 63.673
3.233C50.33 10.957 46.379 33.89 51.995 62.588a170.974 170.974 0 0 0 1.892 8.48c-
3.28.932-6.445 1.924-9.474 2.98C17.309 83.498 0 98.307 0 113.668c0 15.865 18.582 31.778
46.812 41.427a145.52 145.52 0 0 0 6.921 2.165a167.467 167.467 0 0 0-2.01 9.138c-5.354
28.2-1.173 50.591 12.134 58.266c13.744 7.926 36.812-.22 59.273-19.855a145.567 145.567 0
0 0 5.342-4.923a168.064 168.064 0 0 0 6.92 6.314c21.758 18.722 43.246 26.282 56.54
18.586c13.731-7.949 18.194-32.003 12.4-61.268a145.016 145.016 0 0 0-1.535-
6.842c1.62-.48 3.21-.974 4.76-1.488c29.348-9.723 48.443-25.443 48.443-41.52c0-15.417-
17.868-30.326-45.517-39.844Zm-6.365 70.984c-1.4.463-2.836.91-4.3 1.345c-3.24-10.257-
7.612-21.163-12.963-32.432c5.106-11 9.31-21.767 12.459-31.957c2.619.758 5.16 1.557 7.61
2.4c23.69 8.156 38.14 20.213 38.14 29.504c0 9.896-15.606 22.743-40.946 31.14Zm-10.514
20.834c2.562 12.94 2.927 24.64 1.23 33.787c-1.524 8.219-4.59 13.698-8.382 15.893c-8.067
4.67-25.32-1.4-43.927-17.412a156.726 156.726 0 0 1-6.437-5.87c7.214-7.889 14.423-17.06
21.459-27.246c12.376-1.098 24.068-2.894 34.671-5.345a134.17 134.17 0 0 1 1.386
6.193ZM87.276 214.515c-7.882 2.783-14.16 2.863-17.955.675c-8.075-4.657-11.432-22.636-
6.853-46.752a156.923 156.923 0 0 1 1.869-8.499c10.486 2.32 22.093 3.988 34.498
4.994c7.084 9.967 14.501 19.128 21.976 27.15a134.668 134.668 0 0 1-4.877 4.492c-9.933
8.682-19.886 14.842-28.658 17.94ZM50.35 144.747c-12.483-4.267-22.792-9.812-29.858-
15.863c-6.35-5.437-9.555-10.836-9.555-15.216c0-9.322 13.897-21.212 37.076-
29.293c2.813-.98 5.757-1.905 8.812-2.773c3.204 10.42 7.406 21.315 12.477 32.332c-5.137
11.18-9.399 22.249-12.634 32.792a134.718 134.718 0 0 1-6.318-1.979Zm12.378-84.26c-
4.811-24.587-1.616-43.134 6.425-47.789c8.564-4.958 27.502 2.111 47.463 19.835a144.318
144.318 0 0 1 3.841 3.545c-7.438 7.987-14.787 17.08-21.808 26.988c-12.04 1.116-23.565
2.908-34.161 5.309a160.342 160.342 0 0 1-1.76-7.887Zm110.427 27.268a347.8 347.8 0 0 0-
7.785-12.803c8.168 1.033 15.994 2.404 23.343 4.08c-2.206 7.072-4.956 14.465-8.193
22.045a381.151 381.151 0 0 0-7.365-13.322Zm-45.032-43.861c5.044 5.465 10.096 11.566
15.065 18.186a322.04 322.04 0 0 0-30.257-.006c4.974-6.559 10.069-12.652 15.192-
18.18ZM82.802 87.83a323.167 323.167 0 0 0-7.227 13.238c-3.184-7.553-5.909-14.98-8.134-
22.152c7.304-1.634 15.093-2.97 23.209-3.984a321.524 321.524 0 0 0-7.848 12.897Zm8.081
65.352c-8.385-.936-16.291-2.203-23.593-3.793c2.26-7.3 5.045-14.885 8.298-22.6a321.187
321.187 0 0 0 7.257 13.246c2.594 4.48 5.28 8.868 8.038 13.147Zm37.542 31.03c-5.184-
5.592-10.354-11.779-15.403-18.433c4.902.192 9.899.29 14.978.29c5.218 0 10.376-.117
15.453-.343c-4.985 6.774-10.018 12.97-15.028 18.486Zm52.198-57.817c3.422 7.8 6.306
15.345 8.596 22.52c-7.422 1.694-15.436 3.058-23.88 4.071a382.417 382.417 0 0 0 7.859-
13.026a347.403 347.403 0 0 0 7.425-13.565Zm-16.898 8.101a358.557 358.557 0 0 1-12.281
19.815a329.4 329.4 0 0 1-23.444.823c-7.967 0-15.716-.248-23.178-.732a310.202 310.202 0
0 1-12.513-19.846h.001a307.41 307.41 0 0 1-10.923-20.627a310.278 310.278 0 0 1 10.89-
20.637l-.001.001a307.318 307.318 0 0 1 12.413-19.761c7.613-.576 15.42-.876
23.31-.876H128c7.926 0 15.743.303 23.354.883a329.357 329.357 0 0 1 12.335
19.695a358.489 358.489 0 0 1 11.036 20.54a329.472 329.472 0 0 1-11 20.722Zm22.56-
122.124c8.572 4.944 11.906 24.881 6.52 51.026c-.344 1.668-.73 3.367-1.15 5.09c-10.622-
2.452-22.155-4.275-34.23-5.408c-7.034-10.017-14.323-19.124-21.64-27.008a160.789 160.789
0 0 1 5.888-5.4c18.9-16.447 36.564-22.941 44.612-18.3ZM128 90.808c12.625 0 22.86 10.235
22.86 22.86s-10.235 22.86-22.86 22.86s-22.86-10.235-22.86-22.86s10.235-22.86 22.86-
22.86Z"></path></svg>
```

src\interfaces\web\react-app\src\mainjsx

to top

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

src\interfaces\web\react-app\viteconfig.js

to top

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react-swc'

// https://vite.dev/config/
export default defineConfig({
  plugins: [react()],
})
```

src\interfaces\web\services\calendarservice.js

to top

```
/**
 * Calendar Service
 * Integrates with FullCalendar and the backend API to manage workout events
 */

// Remove ES module imports and use the global FullCalendar objects
// These will be loaded via CDN in the HTML
// import { Calendar } from 'https://cdn.jsdelivr.net/npm/@fullcalendar/core/index.js';
// import dayGridPlugin from
'https://cdn.jsdelivr.net/npm/@fullcalendar/daygrid/index.js';
// import timeGridPlugin from
'https://cdn.jsdelivr.net/npm/@fullcalendar/timegrid/index.js';
// import interactionPlugin from
'https://cdn.jsdelivr.net/npm/@fullcalendar/interaction/index.js';

class CalendarService {
  constructor() {
    this.calendar = null;
    this.currentEvents = [];
    this.selectedEvent = null;
    this.isEditMode = false;

    // Initialize when DOM is fully loaded
    document.addEventListener('DOMContentLoaded', () => this.initialize());
  }

  /**
   * Initialize the calendar service and set up event listeners
   */
  initialize() {
    // Setup calendar once DOM is loaded
    this.setupCalendar();

    // Event listeners for the modal
    document.getElementById('createEventBtn')?.addEventListener('click', () =>
this.openCreateEventModal());
    document.getElementById('closeEventModal')?.addEventListener('click', () =>
this.closeEventModal());
    document.getElementById('eventForm')?.addEventListener('submit', (e) =>
this.handleEventFormSubmit(e));
    document.getElementById('isWorkoutSession')?.addEventListener('change', (e) =>
this.toggleWorkoutSessionFields(e));

    // Navigation listener to initialize calendar when tab is clicked
    document.getElementById('navCalendar')?.addEventListener('click', () => {
      // Need to render calendar after display:none is removed
      setTimeout(() => {
        if (this.calendar) {
          this.calendar.render();
        }
      }, 10);
    });
```

```
}

/**
 * Setup FullCalendar instance
 */
setupCalendar() {
  const calendarEl = document.getElementById('calendar');
  if (!calendarEl) return;

  this.calendar = new Calendar(calendarEl, {
    plugins: [dayGridPlugin, timeGridPlugin, interactionPlugin],
    initialView: 'dayGridMonth',
    headerToolbar: {
      left: 'prev,next today',
      center: 'title',
      right: 'dayGridMonth,timeGridWeek,timeGridDay'
    },
    editable: true,
    selectable: true,
    selectMirror: true,
    dayMaxEvents: true,
    events: (fetchInfo, successCallback, failureCallback) => {
      this.fetchEvents(fetchInfo.start, fetchInfo.end)
        .then(events => successCallback(events))
        .catch(error => {
          console.error('Error fetching events:', error);
          failureCallback(error);
        });
    },
    select: (selectInfo) => {
      this.openCreateEventModal(selectInfo);
    },
    eventClick: (clickInfo) => {
      this.handleEventClick(clickInfo.event);
    },
    eventDrop: (dropInfo) => {
      this.handleEventDrop(dropInfo.event);
    },
    eventResize: (resizeInfo) => {
      this.handleEventResize(resizeInfo.event);
    }
  });

  // Initial render
  this.calendar.render();
}

/**
 * Fetch events from the API
 * @param {Date} start Start date
 * @param {Date} end End date
 * @returns {Promise<Array>} Events
 */
```

```
  async fetchEvents(start, end) {
    try {
      const response = await fetch('/api/events');

      if (!response.ok) {
        throw new Error('Failed to fetch events');
      }

      const events = await response.json();

      // Transform events for FullCalendar format
      return events.map(event => ({
        id: event._id,
        title: event.title,
        start: new Date(event.startDate),
        end: new Date(event.endDate),
        description: event.description,
        extendedProps: {
          isWorkoutSession: !!event.workoutSession,
          workoutType: event.workoutSession?.workoutType,
          intensity: event.workoutSession?.intensity,
          videoCallEnabled: event.workoutSession?.videoCallEnabled
        },
        backgroundColor: event.workoutSession ? '#4F46E5' : '#60A5FA',
        borderColor: event.workoutSession ? '#4338CA' : '#3B82F6'
      }));
    } catch (error) {
      console.error('Error fetching events:', error);
      return [];
    }
  }

  /**
   * Open the create event modal with optional initial dates
   * @param {Object} selectInfo Event selection info from FullCalendar
   */
  openCreateEventModal(selectInfo = null) {
    // Reset form
    document.getElementById('eventForm').reset();
    document.getElementById('eventModalTitle').textContent = 'Create Event';
    document.getElementById('workoutSessionFields').classList.add('hidden');

    // Set initial dates if provided
    if (selectInfo) {
      const startInput = document.getElementById('eventStart');
      const endInput = document.getElementById('eventEnd');

      startInput.value = this.formatDateForInput(selectInfo.start);
      endInput.value = this.formatDateForInput(selectInfo.end);
    } else {
      // Default to current time rounded to nearest half hour
      const now = new Date();
      const minutes = Math.ceil(now.getMinutes() / 30) * 30;
```

```
      now.setMinutes(minutes);
      now.setSeconds(0);

      const later = new Date(now);
      later.setHours(later.getHours() + 1);

      document.getElementById('eventStart').value = this.formatDateForInput(now);
      document.getElementById('eventEnd').value = this.formatDateForInput(later);
    }

    // Reset edit mode
    this.isEditMode = false;
    this.selectedEvent = null;

    // Show modal
    document.getElementById('eventModal').classList.remove('hidden');
  }

  /**
   * Open the edit event modal with existing event data
   * @param {Object} event FullCalendar event object
   */
  openEditEventModal(event) {
    // Set form title
    document.getElementById('eventModalTitle').textContent = 'Edit Event';

    // Populate form with event data
    const form = document.getElementById('eventForm');
    form.reset();

    document.getElementById('eventTitle').value = event.title;
    document.getElementById('eventDescription').value = event.extendedProps.description
|| '';
    document.getElementById('eventStart').value = this.formatDateForInput(event.start);
    document.getElementById('eventEnd').value = this.formatDateForInput(event.end);

    // Set workout session fields
    const isWorkoutSession = event.extendedProps.isWorkoutSession;
    document.getElementById('isWorkoutSession').checked = isWorkoutSession;

    if (isWorkoutSession) {
      document.getElementById('workoutSessionFields').classList.remove('hidden');
      document.getElementById('workoutType').value = event.extendedProps.workoutType ||
'other';
      document.getElementById('intensity').value = event.extendedProps.intensity ||
'medium';
      document.getElementById('videoCallEnabled').checked =
event.extendedProps.videoCallEnabled !== false;
    } else {
      document.getElementById('workoutSessionFields').classList.add('hidden');
    }

    // Set edit mode
```

```
    this.isEditMode = true;
    this.selectedEvent = event;

    // Show modal
    document.getElementById('eventModal').classList.remove('hidden');
  }

  /**
   * Close the event modal
   */
  closeEventModal() {
    document.getElementById('eventModal').classList.add('hidden');
  }

  /**
   * Handle form submission for creating or updating events
   * @param {Event} e Form submission event
   */
  async handleEventFormSubmit(e) {
    e.preventDefault();

    const form = e.target;
    const formData = new FormData(form);

    // Create the event data object
    const eventData = {
      title: formData.get('title'),
      description: formData.get('description'),
      startDate: new Date(formData.get('start')).toISOString(),
      endDate: new Date(formData.get('end')).toISOString()
    };

    // Add workout session data if applicable
    if (formData.get('isWorkoutSession')) {
      eventData.workoutSession = {
        workoutType: formData.get('workoutType'),
        intensity: formData.get('intensity'),
        videoCallEnabled: formData.get('videoCallEnabled') === 'on'
      };
    }

    try {
      let response;

      if (this.isEditMode && this.selectedEvent) {
        // Update existing event
        response = await fetch(`/api/events/${this.selectedEvent.id}`, {
          method: 'PUT',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify(eventData)
        });
      } else {
        // Create new event
```

```
        response = await fetch('/api/events', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify(eventData)
        });
      }

      if (!response.ok) {
        throw new Error('Failed to save event');
      }

      // Refresh calendar
      this.calendar.refetchEvents();

      // Close modal
      this.closeEventModal();

    } catch (error) {
      console.error('Error saving event:', error);
      alert('Failed to save event. Please try again.');
    }
  }

  /**
   * Toggle workout session fields based on checkbox state
   * @param {Event} e Change event
   */
  toggleWorkoutSessionFields(e) {
    const workoutSessionFields = document.getElementById('workoutSessionFields');

    if (e.target.checked) {
      workoutSessionFields.classList.remove('hidden');
    } else {
      workoutSessionFields.classList.add('hidden');
    }
  }

  /**
   * Handle event click to view/edit event
   * @param {Object} event FullCalendar event object
   */
  handleEventClick(event) {
    this.openEditEventModal(event);
  }

  /**
   * Handle event drop (dragging to new time/date)
   * @param {Object} event FullCalendar event object
   */
  async handleEventDrop(event) {
    await this.updateEventDates(event);
  }
```

```
  /**
   * Handle event resize
   * @param {Object} event FullCalendar event object
   */
  async handleEventResize(event) {
    await this.updateEventDates(event);
  }

  /**
   * Update event dates after drag or resize
   * @param {Object} event FullCalendar event object
   */
  async updateEventDates(event) {
    try {
      const response = await fetch(`/api/events/${event.id}`, {
        method: 'PUT',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          startDate: event.start.toISOString(),
          endDate: event.end.toISOString()
        })
      });

      if (!response.ok) {
        throw new Error('Failed to update event');
      }

    } catch (error) {
      console.error('Error updating event:', error);
      alert('Failed to update event. Please refresh and try again.');
      this.calendar.refetchEvents();
    }
  }

  /**
   * Format date for datetime-local input
   * @param {Date} date Date to format
   * @returns {string} Formatted date string
   */
  formatDateForInput(date) {
    if (!date) return '';

    // Create a new Date object to avoid modifying the original
    const d = new Date(date);

    // Adjust for timezone if needed
    const offset = d.getTimezoneOffset();
    d.setMinutes(d.getMinutes() - offset);

    // Format to YYYY-MM-DDTHH:MM
    return d.toISOString().slice(0, 16);
  }
```

```
  /**
   * Delete an event
   * @param {string} eventId Event ID
   */
  async deleteEvent(eventId) {
    try {
      const response = await fetch(`/api/events/${eventId}`, {
        method: 'DELETE'
      });

      if (!response.ok) {
        throw new Error('Failed to delete event');
      }

      this.calendar.refetchEvents();
      this.closeEventModal();

    } catch (error) {
      console.error('Error deleting event:', error);
      alert('Failed to delete event. Please try again.');
    }
  }
}

// Initialize the calendar service
const calendarService = new CalendarService();

// Make the service available globally instead of exporting as an ES module
window.calendarService = calendarService;
```

src\interfaces\web\services\meetingservice.js

to top

```
class MeetingService {
    async createMeeting() {
        try {
            const response = await fetch('/api/meetings', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                }
            });
            const data = await response.json();
            if (!response.ok) {
                throw new Error(data.error || 'Error creating meeting');
            }
            return data.meetingId;
        } catch (error) {
            throw new Error('Failed to create meeting: ' + error.message);
        }
    }

    async joinMeeting(meetingId) {
        try {
            const response = await fetch(`/api/meetings/${meetingId}`);
            const data = await response.json();
            if (!response.ok) {
                throw new Error(data.error || 'Error joining meeting');
            }
            return data;
        } catch (error) {
            throw new Error('Failed to join meeting: ' + error.message);
        }
    }
}

// Instead of using ES module export, make it available globally
window.MeetingService = MeetingService;
```

src\interfaces\web\services\messageservice.js

to top

```
/**
 * Message Service
 * Handles real-time messaging between users for workout sessions
 */

class MessageService {
  constructor() {
    this.socket = null;
    this.currentEventId = null;
    this.messageListeners = [];

    // Initialize when DOM is fully loaded
    document.addEventListener('DOMContentLoaded', () => this.initialize());
  }

  /**
   * Initialize the message service
   */
  initialize() {
    // Connect to Socket.IO if available
    if (window.io) {
      this.socket = io();
      this.setupSocketListeners();
    }

    // Set up UI event listeners when on the meetings tab
    document.getElementById('navMeetings')?.addEventListener('click', () => {
      this.loadWorkoutSessions();
    });
  }

  /**
   * Setup Socket.IO event listeners
   */
  setupSocketListeners() {
    if (!this.socket) return;

    // Listen for new messages
    this.socket.on('new-message', (message) => {
      // Notify all registered listeners about the new message
      this.notifyMessageListeners(message);
    });

    // Listen for user joined/left events
    this.socket.on('user-joined', (userData) => {
      console.log(`User joined: ${userData.username}`);
      this.addSystemMessage(`${userData.username} joined the session`);
    });

    this.socket.on('user-left', (userData) => {
      console.log(`User left: ${userData.userId}`);
      this.addSystemMessage(`A user left the session`);
```

```
    });
  }

  /**
   * Join an event chat room
   * @param {string} eventId Event ID to join
   */
  joinEventChat(eventId) {
    if (!this.socket || !eventId) return;

    // Leave current event if any
    if (this.currentEventId) {
      this.socket.emit('leave-event', this.currentEventId);
    }

    // Join new event
    this.currentEventId = eventId;
    this.socket.emit('join-event', eventId);

    // Load previous messages
    this.loadMessages(eventId);
  }

  /**
   * Leave the current event chat room
   */
  leaveEventChat() {
    if (!this.socket || !this.currentEventId) return;

    this.socket.emit('leave-event', this.currentEventId);
    this.currentEventId = null;
  }

  /**
   * Load messages for an event
   * @param {string} eventId Event ID
   * @returns {Promise<Array>} Messages
   */
  async loadMessages(eventId) {
    try {
      const response = await fetch(`/api/messages/event/${eventId}`);

      if (!response.ok) {
        throw new Error('Failed to load messages');
      }

      const messages = await response.json();

      // Update UI with messages
      this.renderMessages(messages);

      return messages;
    } catch (error) {
```

```
      console.error('Error loading messages:', error);
      return [];
    }
  }

  /**
   * Send a message in the current event
   * @param {string} content Message content
   * @returns {Promise<Object>} Sent message
   */
  async sendMessage(content) {
    if (!this.currentEventId || !content.trim()) {
      return null;
    }

    try {
      const response = await fetch('/api/messages', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          content,
          eventId: this.currentEventId
        })
      });

      if (!response.ok) {
        throw new Error('Failed to send message');
      }

      const message = await response.json();
      return message;
    } catch (error) {
      console.error('Error sending message:', error);
      return null;
    }
  }

  /**
   * Add a system message to the chat
   * @param {string} content Message content
   */
  addSystemMessage(content) {
    const systemMessage = {
      content,
      sender: null,
      isSystem: true,
      timestamp: new Date()
    };

    this.notifyMessageListeners(systemMessage);
  }

  /**
```

```
   * Register a message listener
   * @param {Function} listener Callback function to receive messages
   */
  registerMessageListener(listener) {
    if (typeof listener === 'function') {
      this.messageListeners.push(listener);
    }
  }

  /**
   * Unregister a message listener
   * @param {Function} listener Listener to unregister
   */
  unregisterMessageListener(listener) {
    this.messageListeners = this.messageListeners.filter(l => l !== listener);
  }

  /**
   * Notify all registered listeners about a new message
   * @param {Object} message Message object
   */
  notifyMessageListeners(message) {
    this.messageListeners.forEach(listener => {
      try {
        listener(message);
      } catch (error) {
        console.error('Error in message listener:', error);
      }
    });
  }

  /**
   * Load active workout sessions
   */
  async loadWorkoutSessions() {
    try {
      const response = await fetch('/api/events?type=workout');

      if (!response.ok) {
        throw new Error('Failed to load workout sessions');
      }

      const events = await response.json();

      // Filter for upcoming events with workout sessions
      const workoutSessions = events.filter(event =>
        event.workoutSession &&
        new Date(event.endDate) > new Date()
      );

      // Sort by start date (soonest first)
      workoutSessions.sort((a, b) =>
        new Date(a.startDate) - new Date(b.startDate)
```

```
      );

      // Render workout sessions
      this.renderWorkoutSessions(workoutSessions);

      return workoutSessions;
    } catch (error) {
      console.error('Error loading workout sessions:', error);
      return [];
    }
  }

  /**
   * Render workout sessions in the meetings list
   * @param {Array} sessions Workout sessions
   */
  renderWorkoutSessions(sessions) {
    const meetingsList = document.getElementById('meetingsList');
    if (!meetingsList) return;

    // Clear current list
    meetingsList.innerHTML = '';

    if (sessions.length === 0) {
      meetingsList.innerHTML = `
        <div class="text-center py-8 text-gray-500">
          <p>No upcoming workout sessions.</p>
          <p class="mt-2">Create a new workout session in the Calendar tab.</p>
        </div>
      `;
      return;
    }

    // Create session cards
    sessions.forEach(session => {
      const startDate = new Date(session.startDate);
      const endDate = new Date(session.endDate);

      const isActive = startDate <= new Date() && endDate >= new Date();
      const isPast = endDate < new Date();

      const card = document.createElement('div');
      card.className = 'bg-white shadow rounded-lg p-4 border-l-4 border-blue-500';

      card.innerHTML = `
        <div class="flex justify-between items-start">
          <div>
            <h3 class="font-medium text-gray-900">${session.title}</h3>
            <p class="text-sm text-gray-500">
              ${formatDate(startDate)} • ${formatTime(startDate)} -
${formatTime(endDate)}
            </p>
            <p class="text-sm text-gray-700 mt-2">
```

```
              ${session.description || 'No description'}
            </p>
            <div class="mt-2 flex space-x-2">
              <span class="inline-flex items-center px-2 py-0.5 rounded text-xs font-
medium bg-blue-100 text-blue-800">
                ${session.workoutSession.workoutType}
              </span>
              <span class="inline-flex items-center px-2 py-0.5 rounded text-xs font-
medium bg-purple-100 text-purple-800">
                ${session.workoutSession.intensity} intensity
              </span>
            </div>
          </div>
          <div>
            ${isActive ? `
              <button
                class="bg-green-600 text-white py-1 px-3 rounded-md hover:bg-green-700
text-sm"
                data-event-id="${session._id}"
                onclick="app.joinWorkoutSession('${session._id}')"
              >
                Join Now
              </button>
            ` : isPast ? `
              <span class="text-xs text-gray-500">Completed</span>
            ` : `
              <span class="text-xs text-gray-500">Upcoming</span>
            `}
          </div>
        </div>
      `;

      meetingsList.appendChild(card);
    });
  }
}

/**
 * Format date as "Mon, Apr 7"
 * @param {Date} date Date to format
 * @returns {string} Formatted date
 */
function formatDate(date) {
  const options = { weekday: 'short', month: 'short', day: 'numeric' };
  return date.toLocaleDateString('en-US', options);
}

/**
 * Format time as "10:30 AM"
 * @param {Date} date Date to format
 * @returns {string} Formatted time
 */
function formatTime(date) {
```

```
  const options = { hour: 'numeric', minute: '2-digit', hour12: true };
  return date.toLocaleTimeString('en-US', options);
}

// Initialize the message service
const messageService = new MessageService();

// Make the service available globally instead of exporting as an ES module
window.messageService = messageService;
```

src\interfaces\web\services\webrtcservice.js

to top

```
class WebRTCService {
    constructor(socket) {
        this.socket = socket;
        this.peers = {};
        this.peerVideoElements = {};
        this.localStream = null;
        this.currentCamera = null;
    }

    async setupMediaStream() {
        try {
            this.localStream = await navigator.mediaDevices.getUserMedia({
                video: true,
                audio: true
            });
            return this.localStream;
        } catch (error) {
            throw new Error('Error accessing camera and microphone');
        }
    }

    createPeerConnection(userId) {
        const peerConnection = new RTCPeerConnection({
            iceServers: [
                { urls: 'stun:stun.l.google.com:19302' }
            ]
        });

        this.localStream.getTracks().forEach(track => {
            peerConnection.addTrack(track, this.localStream);
        });

        peerConnection.onicecandidate = event => {
            if (event.candidate) {
                this.socket.emit('ice-candidate', event.candidate, userId);
            }
        };

        peerConnection.ontrack = event => {
            if (!this.peerVideoElements[userId]) {
                const videoElement = this.createVideoElement(event.streams[0], false);
                this.peerVideoElements[userId] = videoElement;
                document.getElementById('videoGrid').appendChild(videoElement);
            }
        };

        this.peers[userId] = peerConnection;
        return peerConnection;
    }

    createVideoElement(stream, isLocal) {
        const container = document.createElement('div');
```

```
        container.className = 'video-container';

        const video = document.createElement('video');
        video.srcObject = stream;
        video.autoplay = true;
        if (isLocal) {
            video.muted = true;
            // Keep mirrored view for local video (selfie view)
            video.style.transform = 'scaleX(-1)';
        } else {
            // No mirroring for remote participants
            video.style.transform = 'scaleX(1)';
        }

        container.appendChild(video);
        return container;
    }

    async connectToNewUser(userId) {
        const peerConnection = this.createPeerConnection(userId);
        const offer = await peerConnection.createOffer();
        await peerConnection.setLocalDescription(offer);
        this.socket.emit('offer', offer, userId);
        return peerConnection;
    }

    async handleOffer(offer, senderId) {
        const peerConnection = this.createPeerConnection(senderId);
        await peerConnection.setRemoteDescription(offer);
        const answer = await peerConnection.createAnswer();
        await peerConnection.setLocalDescription(answer);
        this.socket.emit('answer', answer, senderId);
        return peerConnection;
    }

    async handleAnswer(answer, senderId) {
        if (this.peers[senderId]) {
            await this.peers[senderId].setRemoteDescription(answer);
        }
    }

    async handleIceCandidate(candidate, senderId) {
        if (this.peers[senderId]) {
            await this.peers[senderId].addIceCandidate(candidate);
        }
    }

    closePeerConnection(userId) {
        if (this.peers[userId]) {
            this.peers[userId].close();
            delete this.peers[userId];
        }
```

```
        if (this.peerVideoElements[userId]) {
            const videoElement = this.peerVideoElements[userId];
            if (videoElement.parentNode) {
                videoElement.parentNode.removeChild(videoElement);
            }
            delete this.peerVideoElements[userId];
        }
    }

    async toggleVideo() {
        const videoTrack = this.localStream.getVideoTracks()[0];
        if (videoTrack) {
            if (videoTrack.enabled) {
                // If video is currently enabled, stop it completely
                videoTrack.stop();
                videoTrack.enabled = false;
                return false;
            } else {
                // If video is currently disabled, start a new video track
                try {
                    const newStream = await navigator.mediaDevices.getUserMedia({
                        video: this.currentCamera ? { deviceId: { exact:
this.currentCamera } } : true,
                        audio: false
                    });

                    const newVideoTrack = newStream.getVideoTracks()[0];
                    const oldVideoTrack = this.localStream.getVideoTracks()[0];

                    // Replace the video track in the local stream
                    this.localStream.removeTrack(oldVideoTrack);
                    this.localStream.addTrack(newVideoTrack);

                    // Update the video track for all peer connections
                    Object.values(this.peers).forEach(peer => {
                        const senders = peer.getSenders();
                        const videoSender = senders.find(sender => sender.track?.kind
=== 'video');

                        if (videoSender) {
                            videoSender.replaceTrack(newVideoTrack);
                        }
                    });

                    // Update local video element
                    const localVideo = document.querySelector('#videoGrid video');
                    if (localVideo) {
                        localVideo.srcObject = this.localStream;
                    }

                    newVideoTrack.enabled = true;
                    return true;
                } catch (error) {
                    console.error('Error restarting video:', error);
```

```
                    return false;
                }
            }
        }
        return false;
    }

    toggleAudio() {
        const audioTrack = this.localStream.getAudioTracks()[0];
        if (audioTrack) {
            audioTrack.enabled = !audioTrack.enabled;
            return audioTrack.enabled;
        }
        return false;
    }

    cleanup() {
        if (this.localStream) {
            this.localStream.getTracks().forEach(track => track.stop());
        }
        Object.keys(this.peers).forEach(userId => {
            this.closePeerConnection(userId);
        });
    }

    async getAvailableCameras() {
        try {
            const devices = await navigator.mediaDevices.enumerateDevices();
            return devices.filter(device => device.kind === 'videoinput');
        } catch (error) {
            console.error('Error getting cameras:', error);
            return [];
        }
    }

    async switchCamera(deviceId) {
        try {
            // Stop all tracks in the current stream
            if (this.localStream) {
                this.localStream.getTracks().forEach(track => track.stop());
            }

            // Get new stream with selected camera
            this.localStream = await navigator.mediaDevices.getUserMedia({
                video: { deviceId: { exact: deviceId } },
                audio: true
            });

            // Update local video
            const localVideo = document.querySelector('#videoGrid video');
            if (localVideo) {
                localVideo.srcObject = this.localStream;
            }
```

```
            // Update all peer connections with the new stream
            Object.values(this.peers).forEach(peer => {
                const senders = peer.getSenders();
                const videoSender = senders.find(sender => sender.track?.kind ===
'video');
                if (videoSender) {
                    const videoTrack = this.localStream.getVideoTracks()[0];
                    videoSender.replaceTrack(videoTrack);
                }
            });

            this.currentCamera = deviceId;
            return true;
        } catch (error) {
            console.error('Error switching camera:', error);
            throw new Error('Failed to switch camera');
        }
    }
}

// Instead of using ES module export, make it available globally
window.WebRTCService = WebRTCService;
```

src\interfaces\web\shared\scripts\authUtilsjs

to top

```
class AuthUtils {
    static async checkAuthentication() {
        try {
            const response = await fetch('/auth/check', {
                method: 'GET',
                headers: {
                    'Content-Type': 'application/json'
                }
            });

            if (!response.ok) {
                window.location.href = '/auth/login';
                return false;
            }
            return true;
        } catch (error) {
            console.error('Authentication check failed:', error);
            window.location.href = '/auth/login';
            return false;
        }
    }
}

export default AuthUtils;
```

webrtc-implementationmd

to top

WebRTC Implementation in Workoutmate

Architecture Overview

The WebRTC implementation in Workoutmate uses a peer-to-peer connection model with Socket.IO for signaling.

Key Components

1. **WebRTCService**: Manages peer connections, media streams, and WebRTC signaling
2. **Socket.IO**: Handles signaling between peers
3. **App**: Coordinates authentication, UI, and initialization

Signaling Flow

1. User A joins a meeting room (via Socket.IO)
2. User A's presence is announced to all room members
3. For each existing member, User A creates a peer connection and sends an offer
4. Existing members receive the offer and respond with an answer

5. ICE candidates are exchanged between peers
6. Media streams are connected when all signaling is complete

Media Stream Handling

- Local media stream is obtained via `navigator.mediaDevices.getUserMedia()`
- Remote streams are attached to video elements when received
- Camera and microphone controls toggle tracks on the local stream

Critical Components

Peer Connection Creation

```
createPeerConnection(userId) {
    const peerConnection = new RTCPeerConnection({
        iceServers: [
            { urls: 'stun:stun.l.google.com:19302' }
        ]
    });

    this.localStream.getTracks().forEach(track => {
        peerConnection.addTrack(track, this.localStream);
    });

    // Handle ICE candidate generation
    // Handle incoming tracks

    this.peers[userId] = peerConnection;
    return peerConnection;
}
```

Signaling Events

- `offer`: Initiates a connection
- `answer`: Responds to an offer
- `ice-candidate`: Exchanges network connection information
- `user-joined`: Notifies when a new user enters the room
- `user-left`: Notifies when a user leaves the room

Testing Considerations

- Ensure ICE servers are correctly configured
- Verify media permissions are requested and managed properly
- Test connection establishment in various network conditions
- Validate camera and microphone toggle functionality