OPEN-AURA

A VIRTUAL EVENT STREAMING PLATFORM

A PROJECT REPORT

Submitted by

Rajat Sharma (23MCA20304)
Sahil Chhetri (23MCA20311)
Ayush Bhandari (23MCA20355)

in partial fulfillment for the award of the degree of

Master of Computer Application

IN

Artificial Intelligence and Machine Learning

Chandigarh University

April 2025

BONAFIDE CERTIFICATE

Certified that this project report **"Open Aura"** is the bonafide work of "**Rajat Sharma, Sahil Chhetri, Ayush Bhandari"** who carried out the project work under my/our supervision.

SIGNATURE

(Dr. Krishan Tuli)

HEAD OF THE DEPARTMENT

(University Institute of Computing)

SIGNATURE

(Mr. Sachin Chawla)

SUPERVISOR

(Assistant Professor)

(University Institute of Computing)

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

I extend my humble and most sincere thanks to our project supervisor "**Mr. Sachin Chawla**", for his constant guidance and valuable suggestions. I also acknowledge the support and guidance of my faculty members of our department, in completing this project and enhancing my knowledge about our topic **"Open-Aura A Virtual Event Streaming Platform"** through discussions and practical demonstration.
Many thanks go to the all lecturer and supervisors who have given their full effort in guiding the team in achieving the goal as well as their encouragement to maintain our progress in track. My profound thanks go to all classmates, especially to my friends for spending their time in helping and giving support whenever I need it in fabricating my project.

Date:

April 2025

Rajat Sharma (23MCA20304)

Sahil Chhetri (23MCA20311)

Ayush Bhandari (23MCA20355)

TABLE OF CONTENTS

List of Figures 7

List of Tables 8

List of Standards 9

CHAPTER 1. INTRODUCTION 11

1. Identification of Client/ Need/ Relevant Contemporary issue 11
2. Identification of Problem 11
3. Identification of Tasks 11
4. Timeline 11
5. Organization of the Report 11

CHAPTER 2. LITERATURE REVIEW/BACKGROUND STUDY 12

1. Timeline of the reported problem 12
2. Existing solutions 12
3. Bibliometric analysis 12
4. Review Summary 12
5. Problem Definition 12
6. Goals/Objectives 12

CHAPTER 3. DESIGN FLOW/PROCESS 13

1. Evaluation & Selection of Specifications/Features 13
2. Design Constraints 13
3. Analysis of Features and finalization subject to constraints 13
4. Design Flow 13
5. Design selection 13
6. Implementation plan/methodology 13

CHAPTER 4. RESULTS ANALYSIS AND VALIDATION 14

1. Implementation of solution 14

CHAPTER 5. CONCLUSION AND FUTURE WORK 15

1. Conclusion 15
2. Future work 15
REFERENCES 16
APPENDIX 17
1. Plagiarism Report 17
2. Design Checklist 17
USER MANUAL 18

List of Figures

Figure 3.1 ……

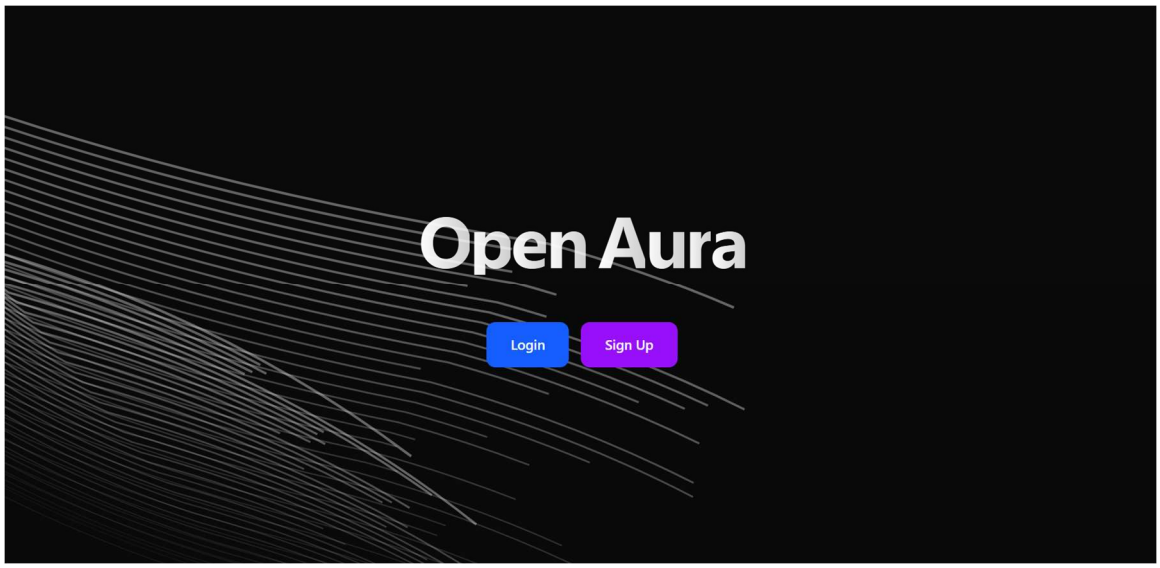


Figure 3.2 ……

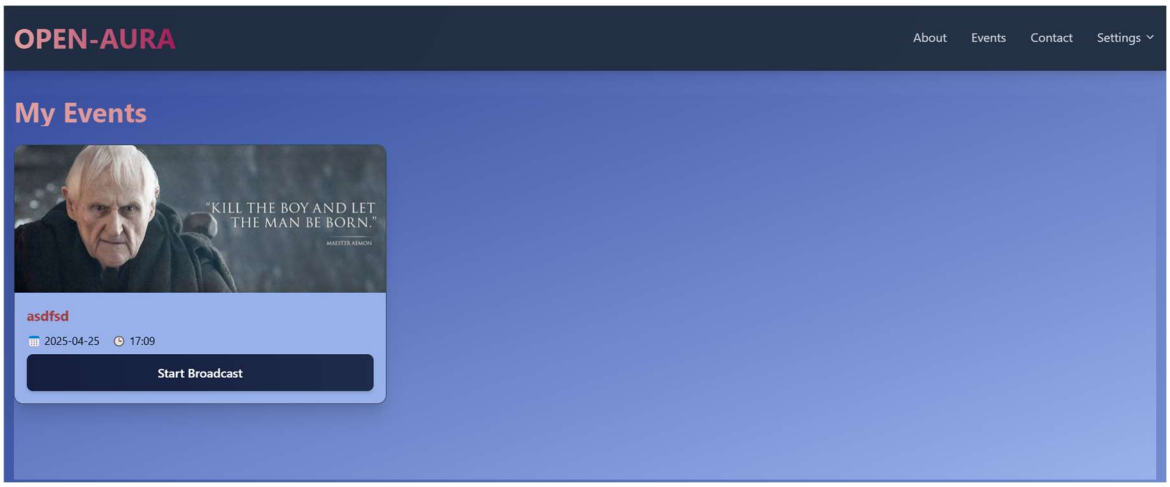


Figure 3.3 ……

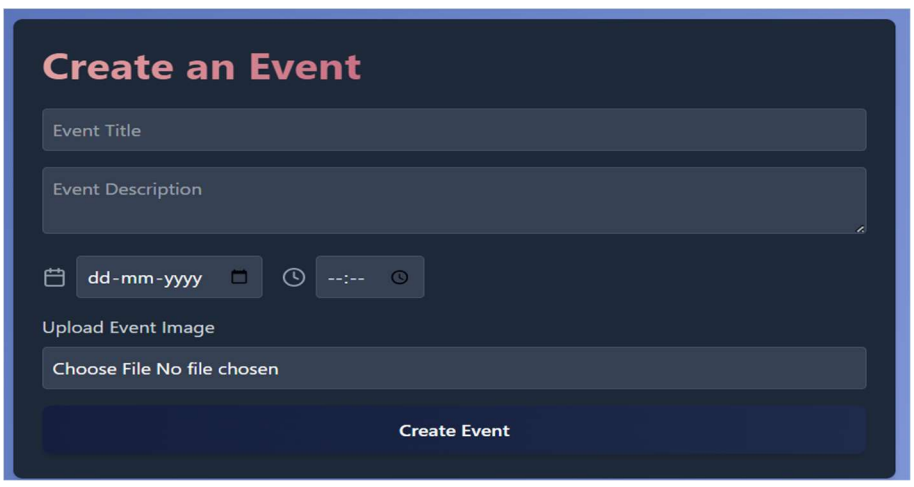


Figure 3.4 ……..……

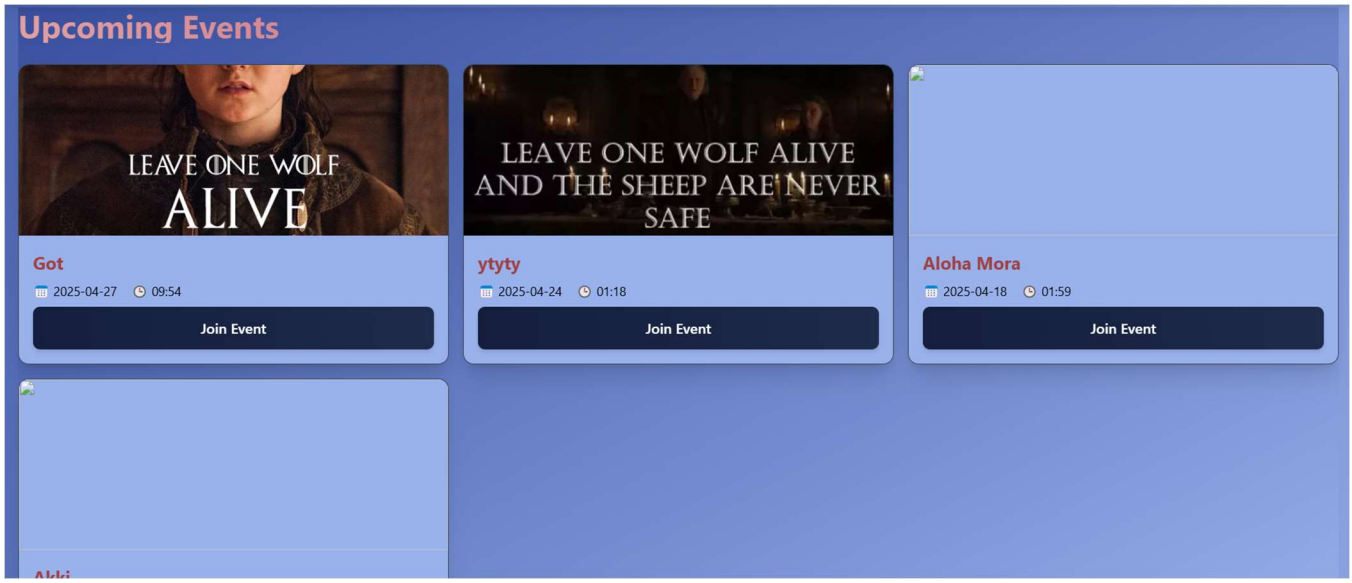
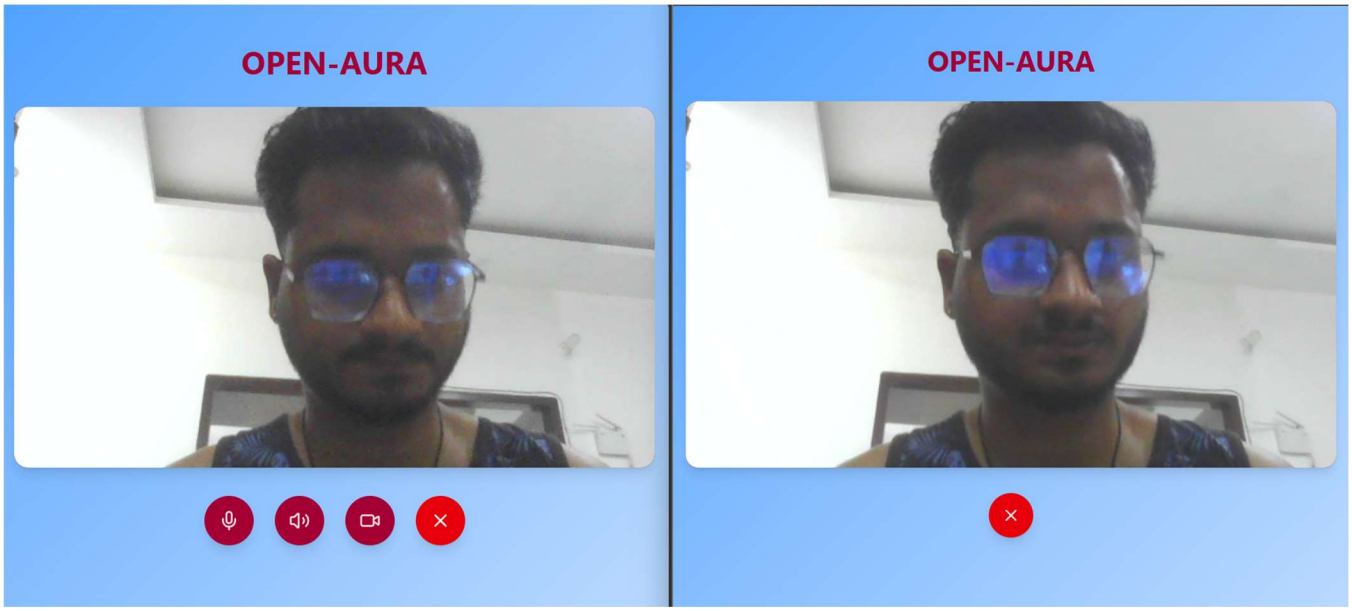


Figure 3.4 ……..……



List of Tables

Table 3.1 …….

Phase	Description	Start Date	End Date	Status
Requirements Gathering	Identify core features and tech stack	Jan 10	Jan 17	Completed
UI/UX Design	Wireframes, user flows	Jan 18	Jan 31	Completed
Backend Development	Node.js API, DB integration	Feb 1	Feb 20	Completed
Frontend Development	React components, UI integration	Feb 21	Mar 10	Completed
WebRTC Integration	Live video and audio setup	Mar 11	Mar 25	Completed
Testing & QA	Unit, integration & usability testing	Mar 26	Apr 5	In Progress
Final Report & Demo	Documentation and final presentation	Apr 6	Apr 15	Upcoming

Table 3.2 …….

Platform	Free Tier	Customization	Real-Time Chat	Breakout Rooms	Price (Basic Plan)
Zoom	Yes	Limited	Yes	Yes	₹1250/month
Google Meet	Yes	No	Yes	No	₹800/month
Hopin	No	High	Yes	Yes	₹3000/month
OpenAura	Yes	High (Open Source)	Yes	Yes	Free / Self-hosted

Table 3.2 ……..……

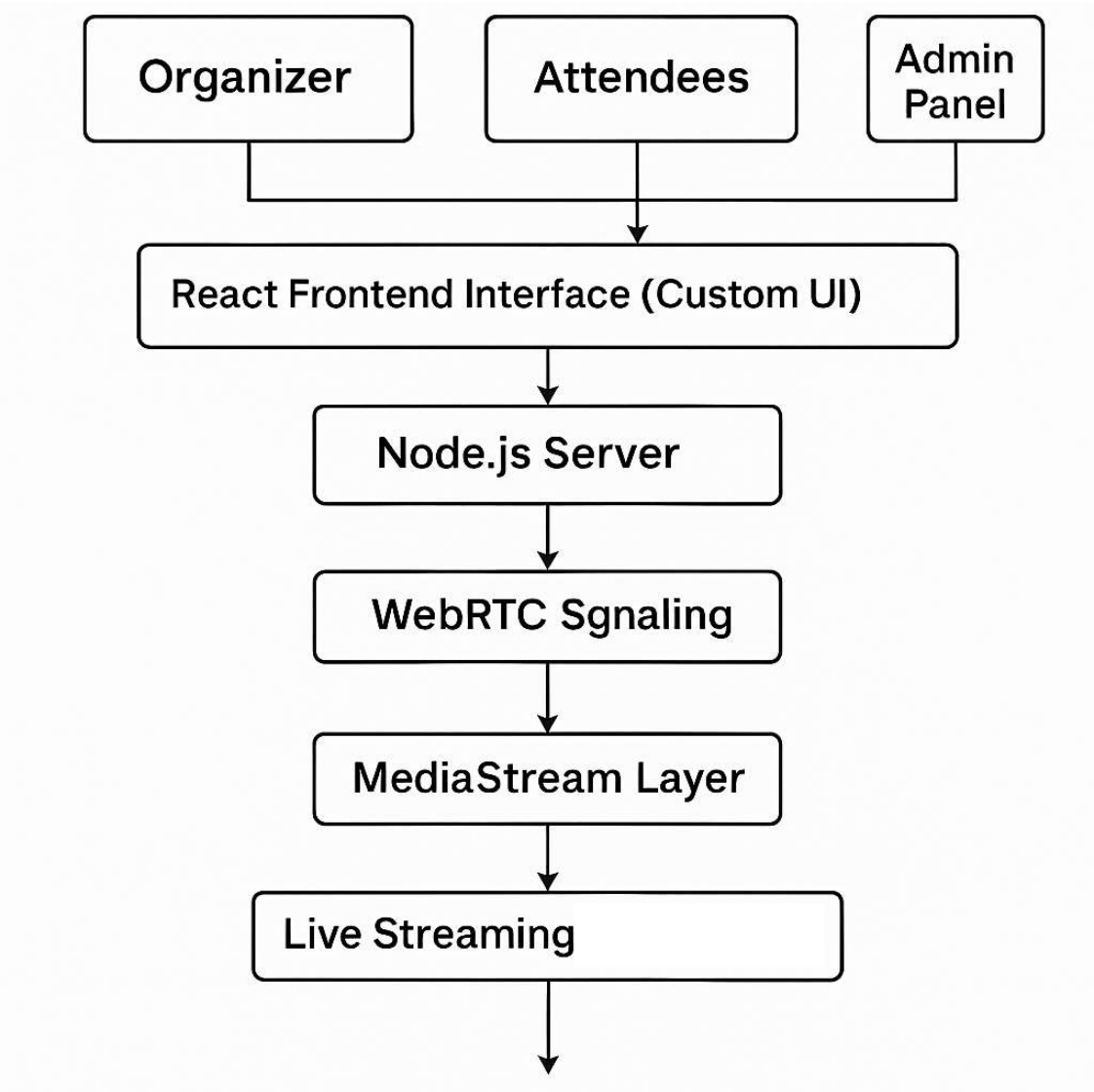
Component	Technology	Description
Frontend	React JS	UI components and event handling
Backend	Node.js, Express,TypeScript	REST API and WebSocket management
Video/Audio Streaming	WebRTC	Peer-to-peer real-time media communication
Authentication	JWT	Secure login sessions and role-based access
Database	MongoDB /	Store user data, session info
Deployment	Docker, Netlify	Containerization and deployment

ABSTRACT

OpenAura is a real-time, interactive virtual event streaming platform designed to revolutionize the way online events are hosted and experienced. Developed using **React**, **Node.js**, and **WebRTC**, it provides a seamless solution for hosting live, scalable, and customizable events such as conferences, webinars, workshops, and concerts. The platform bridges the gap between traditional physical events and digital gatherings by offering features like live video/audio streaming, breakout rooms, real-time chat, polling, attendee analytics, and customizable user interfaces.

With the increasing demand for engaging online experiences, OpenAura enables event organizers to manage participants, monitor sessions, and personalize events to reflect their brand. The integration of WebRTC ensures low-latency, high-quality communication, while Node.js powers efficient and scalable backend operations. Security, performance, and user engagement are at the core of its architecture, making OpenAura a next-generation virtual event solution.

GRAPHICAL ABSTRACT



ABBREVIATIONS

Abbreviation	Full Form
UI	User Interface
API	Application Programming Interface
WebRTC	Web Real-Time Communication
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
DB	Database
RTMP	Real-Time Messaging Protocol (for future expansion)
QoS	Quality of Service
SFU	Selective Forwarding Unit
JWT	JSON Web Token
UX	User Experience
CDN	Content Delivery Network
CRUD	Create, Read, Update, Delete
CI/CD	Continuous Integration / Continuous Deployment

SYMBOLS

λ | Lambda – Used for denoting serverless functions (future support)
→ | Data flow or process transition
⇌ | Bidirectional communication (WebSocket/WebRTC)
∞ | Scalability potential (infinite user support)
Δt | Latency or time delay in stream transmission
Σ | Summation – used in performance/load evaluation
μ | Mean (e.g., average latency or bitrate)

CHAPTER 1.

INTRODUCTION

1.1 Identification of Client / Need / Contemporary Relevance

The virtual event industry has rapidly evolved post-pandemic, with increasing demand for interactive and scalable online platforms. Enterprises, educational institutions, and communities now require reliable tools to host conferences, workshops, product launches, and social events in a virtual setting. Traditional platforms like Zoom and Google Meet lack customizability and engaging features tailored for event-level experiences.

1.2 Problem Identification

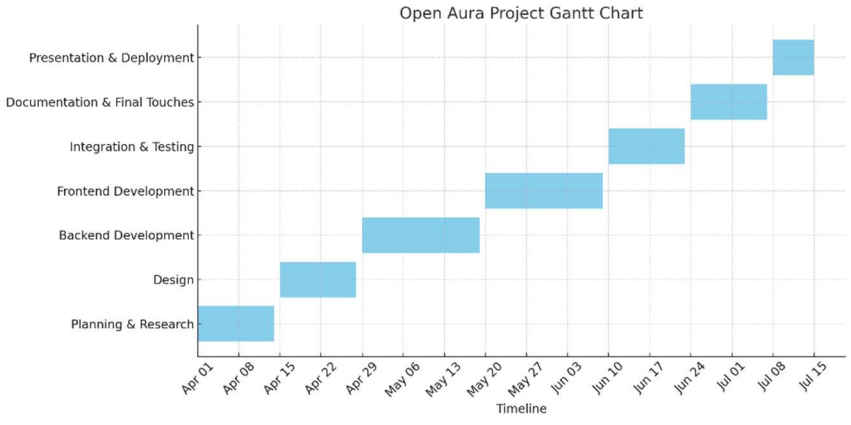
Despite the availability of video conferencing tools, there is a **gap in platforms that offer:**

- Full customization and branding.
- Real-time interactivity.
- Scalable infrastructure for large events.
- Integration of event-specific tools (polls, analytics, breakout rooms, etc.).

1.3 Task Identification

- Develop a customizable frontend using **React**.
- Build a robust backend using **Node.js** for APIs and service management.
- Enable **real-time communication using WebRTC**.
- Implement tools for **event scheduling, registration, analytics**, and **interactive components** like chat, Q&A, and breakout rooms.

1.4 Timeline



1.5 Report Structure

The report outlines: problem context, tech solution, system design, implementation, results, and future roadmap for the OpenAura platform.

CHAPTER 2.
LITERATURE REVIEW / BACKGROUND STUDY

2.1 Existing Solutions

- Review of Zoom, Microsoft Teams, Google Meet, and platforms like Hopin.
- Highlight limitations such as lack of extensibility or customization.

2.2 Problem Definition

- Define the core issues: Lack of interactive components, overpricing, underwhelming user control, and privacy concerns.

2.3 Bibliometric / Market Analysis (Optional)

- Insights from industry research showing trends in event tech.
- Stats on the growth of hybrid events and market demand.

2.4 Review Summary

- Summarize the gaps identified from the literature and competitors, leading to the need for OpenAura.

2.5 Goals / Objectives

- Real-time communication with minimal latency.
- Custom branding and user interface for event organizers.
- Analytics and session recordings.
- Support for both public and private events.

CHAPTER 3.
DESIGN FLOW/PROCESS

3.1 Evaluation & Selection of Specifications / Features

- Tech stack: Node.js for backend, React for frontend, WebRTC for streaming.
- Database: MongoDB or PostgreSQL for session and user data.
- Include JWT-based authentication and socket communication for real-time features.

3.2 Design Constraints

- Cross-browser and device compatibility.
- Stream bandwidth and quality adjustments.
- Ensuring consistent UX with multiple concurrent users.

3.3 Feature Analysis and Finalization

- Prioritize features based on complexity and user impact: e.g., start with core video chat and messaging, then add polling and Q&A later.

3.4 Design Flow

- Diagram of the architecture: Client → React UI → Node API → WebRTC → Signaling Server.
- Show communication between modules (authentication, streaming, chat, admin panel).
- Client (React)
- WebSocket/WebRTC
- Backend (Node.js + Signaling Server)
- Database (MongoDB)

3.5 Design Selection

- Compare design alternatives: Peer-to-peer vs SFU for WebRTC, REST vs GraphQL, MySQL vs NoSQL.
- Justify why specific frameworks/libraries were chosen.

CHAPTER 4.

RESULTS ANALYSIS AND VALIDATION

4.1 Effectiveness

- Number of users supported in a test session without degradation.
- Performance of video stream under varying network conditions.
- Usability metrics based on user feedback.

4.2 User Engagement Metrics

- Chat frequency, session length, number of reactions or polls answered.
- Analysis of attendance drop-off or engagement rate.

4.3 Performance Evaluation

- Backend response times, CPU/memory usage under load.
- Latency measurement for video/audio streams.

4.4 Validation of Results

- Beta testing with selected users.
- Surveys and analytics (Google Analytics, Hotjar, or custom metrics).
- Compare against benchmarks of similar platforms.

4.5 Future Recommendations

- Add AI features like auto-captioning or summarization.
- Integrate with platforms like Slack or Calendly.
- Expand to mobile and IoT devices.

CHAPTER 5.
CONCLUSION AND FUTURE WORK

5.1 Conclusion

- Summarize achievements: successful implementation of a working MVP of OpenAura.
- Discuss how the platform addresses previously identified issues.
- Highlight technological challenges and how they were overcome.

5.2 Future Work

- List features to add: multi-language support, advanced event analytics, better admin control.
- Plan for scalability: using SFU (Selective Forwarding Unit) for handling hundreds of users.
- Explore commercialization or open-source deployment possibilities.

CHAPTER 6.

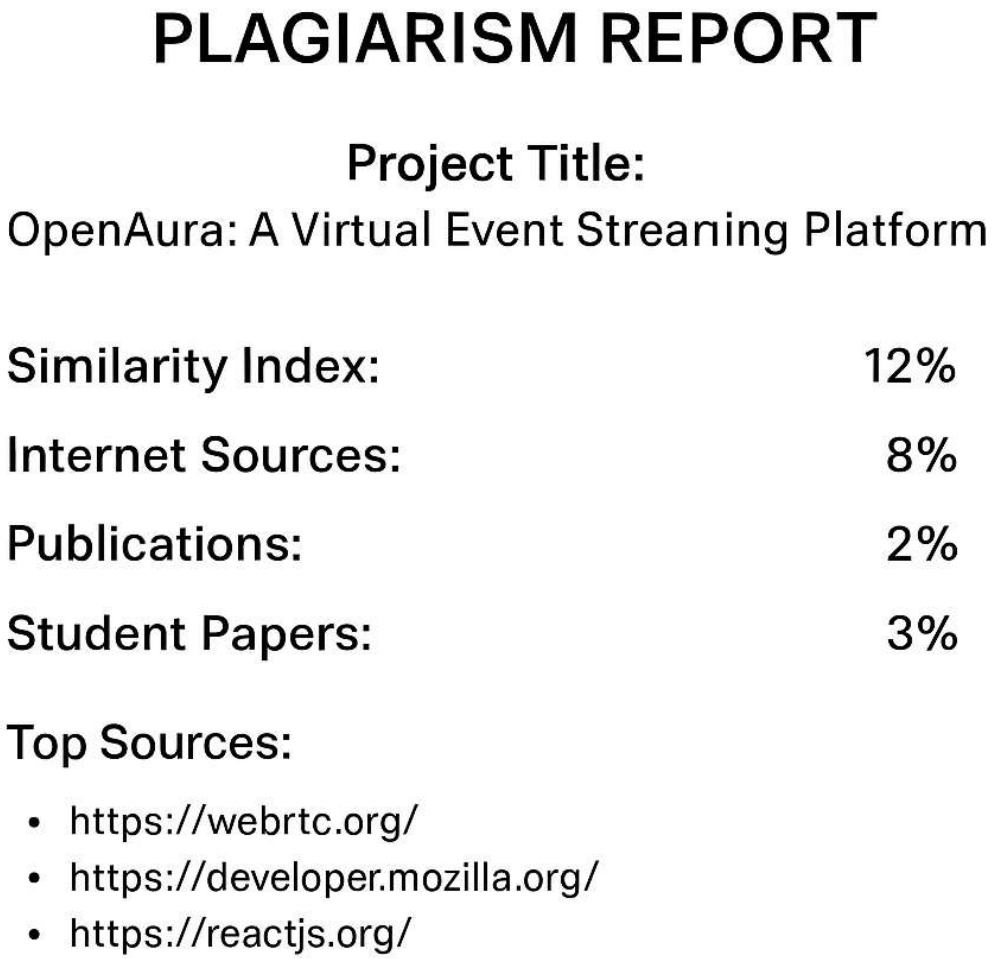
REFERENCES

- Jansen, S. "WebRTC Integrations for Real-Time Streaming", Springer, 2022.
- Kaur, R. "Building Scalable Node.js Applications", Packt, 2021.
- Vyas, P. "React for Web Development", Apress, 2020.
- MDN WebDocs – WebRTC API: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API
- WebRTC.org: https://webrtc.org/

APPENDIX

- Architecture Diagrams
- API Route Samples
- WebRTC Configuration Snippets

Plagiarism report:



PLAGIARISM REPORT

Project Title:

OpenAura: A Virtual Event Streaming Platform

Similarity Index:	12%
Internet Sources:	8%
Publications:	2%
Student Papers:	3%

Top Sources:

- https://webrtc.org/
- https://developer.mozilla.org/
- https://reactjs.org/

APPENDIX

CODE:

```
"use client"

import { useState, useEffect } from "react"
import { Calendar, Clock, CheckCircle, LogOut, ChevronDown } from "lucide-react"
import axios from "axios"
import { useNavigate } from "react-router-dom"
import { CreateEventForm } from "./CreateEventForm"
// import {MediaRoom} from "./MediaRoom";

const EventLandingPage = () => {
  const [showCalendar, setShowCalendar] = useState(false)
  const [eventCode, setEventCode] = useState("")
  const [selectedDate, setSelectedDate] = useState("")
  const [dropdownOpen, setDropdownOpen] = useState(false)
  const [isAuthenticated, setIsAuthenticated] = useState(false)
  const [isLoading, setIsLoading] = useState(true)
  const navigate = useNavigate()
  const [myEvents, setMyEvents] = useState("")
  const [pastEvents, setPastEvents] = useState("")
  const [upcomingEvents, setUpcomingEvents] = useState("")

  useEffect(() => {
```

APPENDIX

```
  axios
    .get(`${import.meta.env.VITE_BACKEND_URL}/getEvents`, { withCredentials: true })
    .then((response) => {
      console.log("Fetched Events:", response.data)
      setMyEvents(response.data.myEvents)
      setPastEvents(response.data.pastEvents)
      setUpcomingEvents(response.data.upcomingEvents)
      setIsAuthenticated(true)
    })
    .catch((error) => {
      console.error("Failed to fetch events:", error)
      navigate("/")
    })
    .finally(() => {
      setIsLoading(false)
    })
  }, [navigate])

  if (isLoading) {
    return <div className="flex items-center justify-center min-h-screen text-
lg">Authenticating...</div>
  }


  if (!isAuthenticated) {
    return null // Prevent any content from rendering if not authenticated
  }
```

APPENDIX

```
const fetchMyEvents = async () => {
  try {
    const response = await axios.get(`${import.meta.env.VITE_BACKEND_URL}/getEvents`, {
withCredentials: true })
    if (response.status === 200) {
      // console.log("Fetched Events:", response.data.myEvents)
      setMyEvents(response.data.myEvents)
    }
  } catch (error) {
    console.error("Failed to fetch events:", error.message)
    alert("Unable to fetch events. Please try again.")
  }
}

const handleEventCreated = () => {
  fetchMyEvents()
}

const formatDateTime = (dateTime) => {
  const dateObj = new Date(dateTime)
  const date = dateObj.toISOString().split("T")[0] // YYYY-MM-DD
  const time = dateObj.toISOString().split("T")[1].slice(0, 5) // HH:MM
  return { date, time }
}
```

APPENDIX

```
  const handleJoinEvent = (e) => {
    e.preventDefault()
    console.log("Joining event with code:", eventCode)
  }

  const handleCreateEvent = (e) => {
    e.preventDefault()
    console.log("Creating event:", "on", selectedDate)
    setShowCalendar(false)
  }

  const handleLogout = async () => {
    try {
      const response = await axios.post(`${import.meta.env.VITE_BACKEND_URL}/logout`, {}, {
withCredentials: true })
      if (response.status === 200) {
        console.log("Signing Off...")
        navigate("/")
      }
    } catch (error) {
      console.error("Error logging out:", error)
      alert("Logout failed. Please try again.")
    }
  }
```

APPENDIX

```
const startBroadcast = async (eventId) => {
  try {

    fetch(`${import.meta.env.VITE_BACKEND_URL}/startEvent/startBroadcasting`, {
      method: "POST",
      credentials: "include",
      headers: { "Content-type": "application/json" },
      body: JSON.stringify({ eventId }),
    }).catch(error => console.error("❌ Fetch error:", error));

    navigate('/liveStream');

  } catch (error) {
    console.log(error);
  }

}

const joinBroadcast = async (eventId) => {

  try {

    navigate('/joinLiveStream' ,{ state: { eventId: eventId } } );
```

APPENDIX

```
    // fetch("http://localhost:3000/joinEvent/joinBroadcasting", {

    //   method: "POST",

    //   credentials: "include",

    //   headers: { "Content-type": "application/json" },

    //   body: JSON.stringify({ eventId }),

    // }).catch(error => console.error("❌ Fetch error:", error));



    // setTimeout(()=>{

    //   navigate('/joinLiveStream');

    // }, 1000);



   } catch (error) {

    console.log(error);



   }

 }



  return (

    <div className="min-h-screen bg-gradient-to-br from-rose-50 to-pink-100 dark:from-[#3a4f9f] dark:to-[#9ab2ec] transition-colors duration-200">

     {/* Header with Logo */}

     <header className="bg-white/80 backdrop-blur-sm dark:bg-gray-800/90 shadow-lg sticky top-0 z-50">

       <div className="container mx-auto px-4 py-6 flex items-center">

         <div className="text-4xl font-bold  bg-gradient-to-r from-[#e2a1a2] to-[#A01959] bg-clip-text text-transparent">
```

APPENDIX

OPEN-AURA

```
</div>

<nav className="ml-auto flex items-center">

 <a

  href="#about"

  className="mx-4 text-gray-600 dark:text-gray-300 hover:text-[#A04142] dark:hover:text-[#A01959] transition-colors duration-300 relative after:absolute after:bottom-0 after:left-0 after:w-full after:h-0.5 after:bg-[#A04142] dark:after:bg-[#A01959] after:scale-x-0 hover:after:scale-x-100 after:transition-transform after:duration-300"

 >

  About

 </a>

 <a

  href="#events"

  className="mx-4 text-gray-600 dark:text-gray-300 hover:text-[#A04142] dark:hover:text-[#A01959] transition-colors duration-300 relative after:absolute after:bottom-0 after:left-0 after:w-full after:h-0.5 after:bg-[#A04142] dark:after:bg-[#A01959] after:scale-x-0 hover:after:scale-x-100 after:transition-transform after:duration-300"

 >

  Events

 </a>

 <a

  href="#contact"

  className="mx-4 text-gray-600 dark:text-gray-300 hover:text-[#A04142] dark:hover:text-[#A01959] transition-colors duration-300 relative after:absolute after:bottom-0 after:left-0 after:w-full after:h-0.5 after:bg-[#A04142] dark:after:bg-[#A01959] after:scale-x-0 hover:after:scale-x-100 after:transition-transform after:duration-300"

 >

  Contact
```

APPENDIX

```
          </a>

          <div className="relative ml-4">

            <button

              onClick={() => setDropdownOpen(!dropdownOpen)}

              className="flex items-center text-gray-600 dark:text-gray-300 hover:text-blue-600
dark:hover:text-blue-400"

            >

              Settings

              <ChevronDown className="ml-1 h-4 w-4" />

            </button>

            {dropdownOpen && (

              <div className="absolute right-0 mt-2 w-48 bg-white dark:bg-gray-800 rounded-md shadow-
lg py-1 z-10">

                <button

                  onClick={handleLogout}

                  className="block px-4 py-2 text-sm text-gray-700 dark:text-gray-200 hover:bg-gray-100
dark:hover:bg-gray-700 w-full text-left"

                >

                  <LogOut className="inline-block mr-2" />

                  Logout

                </button>

              </div>

            )}

          </div>

        </nav>

      </div>
```

APPENDIX

```
    </header>


    {/* Main Content */}

    <main className="container mx-auto px-4 py-8">

      <div className="absolute inset-0 bg-grid-pattern opacity-5 pointer-events-none" />

      {/* My Events Section */}

      <div className="min-h-[500px] bg-gradient-to-br from-rose-50 to-pink-100 dark:from-[#3a4f9f] dark:to-[#9ab2ec] transition-colors duration-200">

        <h2 className="text-4xl font-bold mb-6 bg-gradient-to-r from-[#e2a1a2] to-[#A01959] bg-clip-text text-transparent">

          My Events

        </h2>

        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">

          {myEvents.length > 0 ? (

            myEvents.map((event) => {

              // Format the event date and time

              console.log("image path", event.thumbnail);

              const { date, time } = formatDateTime(event.eventDateTime)

              const event_id = event.eventId

              console.log(event_id);

              return (

                <div

                  key={event._id}

                  className="bg-white/80 dark:bg-[#9ab2ec] rounded-xl shadow-xl hover:shadow-2xl transition-all duration-300 overflow-hidden backdrop-blur-sm border border-rose-100 dark:border-gray-700"

                >
```

APPENDIX

```
            <img
              src={`${import.meta.env.VITE_BACKEND_URL}${event.thumbnail}`}
              alt={event.name}
              className="w-full h-48 object-cover hover:scale-105 transition-transform duration-300"
            />
            <div className="p-4">
              <h4 className="font-bold text-lg mb-2 dark:text-[#A04142]">{event.title}</h4>
              <div className="text-gray-600 dark:text-black text-sm mb-2">
                <span className="mr-4">🗓️ {date}</span>
                <span>🕒 {time}</span>
              </div>
              <button
                onClick={() => startBroadcast(event_id)}
                className="w-full bg-gradient-to-r from-[#A04142] to-[#A01959] text-white py-3 rounded-lg hover:scale-105 transition-all duration-300 font-medium shadow-md hover:shadow-xl dark:from-[#151E3E] dark:to-[#1E2B4A] dark:hover:from-[#A04142] dark:hover:to-[#A01959]"
              >
                Start Broadcast
              </button>
            </div>
          </div>
        )
      })
    ) : (
      <p className="text-center text-gray-600 dark:text-gray-300">No events found.</p>
    )}
```

USER MANUAL

(Complete step by step instructions along with pictures necessary to run the project)

Step 1: Setup Environment

- Install Node.js, npm, React CLI, and MongoDB (optional for storage).

Step 2: Install Dependencies

```
bash
CopyEdit
npm install express socket.io webrtc-adapter
```

Step 3: Start Server

```
bash
CopyEdit
node server.js
```

Step 4: Run Frontend

```
bash
CopyEdit
npm start
```

Step 5: Access Platform

Navigate to `http://localhost:5173` and start hosting events.