

Title
Video-Conferencing Web-App using twilio
A

Project Report
*submitted in partial fulfillment of the
requirements for the award of the degree of*

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING
with

Specialization in CCVT

by

SAP ID

Roll No.

Name

500087620

R2142201876

Ashutosh Lodha

under the guidance of
Mr. Saurabh Shanu
Assistant Professor - Senior Scale,
Systemics Cluster
School of Computer Science



Systemics Cluster
School of Computer Science
University of Petroleum & Energy Studies
Bidholi, Via Prem Nagar, Dehradun, UK
April – 2023



CANDIDATE'S DECLARATION

We hereby certify that the project work entitled “**Video-Conferencing Web-App using twilio**” in partial fulfilment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in CCVT, and submitted to the Systemics Cluster at School of Computer Science, University of Petroleum & Energy Studies, Dehradun, is an authentic record of my work carried out during the period from **January, 2023** to **April, 2023** under the supervision of **Mr. Saurabh Shanu**, Assistant Professor - Senior Scale, Systemics Cluster, School of Computer Science.

The matter presented in this project has not been submitted by me for the award of any other degree of this or any other University.

Name: Ashutosh Lodha

SAP ID: 500087620

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 23rd April, 2023

Mr. Saurabh Shanu
(Subject Faculty)



School of Computer Science

University of Petroleum & Energy Studies, Dehradun

Project Based Learning:

I

PROJECT TITLE:

ABSTRACT

The aim of this project is to develop a web-based video conferencing application using **Twilio API**. The project implements real-time communication between multiple users through video, audio, and text chat. The web application is developed using **React.js** for the frontend and **Node.js** for the backend. The project also makes use of the Redux Toolkit for state management and **Axios** for HTTP requests. The Twilio Video API is used for handling video conferencing features such as participant management, video and audio tracks, and room creation and joining.

The report discusses the challenges faced in implementing the video conferencing application, including the need for reliable and scalable infrastructure to handle real-time communication. The report also explores the advantages of using cloud services, specifically Heroku, for deploying and hosting the web application. The deployment process is detailed in the report, including the configuration of environment variables and the setup of the Twilio API keys.

The results and discussions section of the report presents a detailed analysis of the performance of the web application. The report includes discussions on the limitations of the application, such as browser compatibility issues, and potential future improvements, such as implementing screen sharing and file transfer functionalities.

In conclusion, the project demonstrates the feasibility of developing a web-based video conferencing application using Twilio API. The use of cloud services, specifically **Heroku**, allowed for easy deployment and scalability of the application. The project's methodology, algorithm, and implementation analysis provide valuable insights into the development of real-time communication applications using web technologies..

Keywords: Twilio API, React.js, Node.js, Axios, Heroku Cloud

Contents

INTRODUCTION (6-7 pages).....	1
PROBLEM STATEMENT (1 page)	13
OBJECTIVE (1 page).....	14
RELATED WORK (6 pages).....	15
WHY DO WE NEED CLOUD FOR THE PROJECT (5-6 PAGES)	20
METHODOLOGY (4-5 pages).....	23
ALGORITHM (2-3 pages)	27
RESULT AND DISCUSSIONS (8-10 pages)	29
PUBLIC CLOUD DEPLOYMENT (4-5 pages).....	32
SCREENSHOTS.....	36
Key Bibliography/References	38

INTRODUCTION

In recent times, video-conferencing has become an essential mode of communication for individuals and businesses alike. With the increasing need for remote work and virtual communication, the demand for a reliable and efficient video-conferencing web application has increased tremendously. Our project aims to develop a video-conferencing web application using Twilio, a cloud communications platform, and deploy it on a public cloud platform.

The main goal of the project is to provide users with a seamless and hassle-free video-conferencing experience that is accessible from any device and location. The web application will allow users to create and join virtual meeting rooms, share screens, and communicate via audio and video in real-time. To ensure scalability and reliability, the application will be deployed on a public cloud platform.

In this project, we have utilized the Twilio API, which provides an easy-to-use interface for building voice, video, and messaging applications. The Twilio API will allow us to create and manage video rooms, provide secure authentication for users, and manage audio and video quality. We have also chosen to deploy our application on a public cloud platform to ensure that it can handle large traffic volumes, provide high availability, and offer scalability and flexibility.

The remainder of this report is organized as follows: Section 2 outlines the problem statement and the key objectives of the project. Section 3 discusses the methodology used to develop the video-conferencing web application, while Section 4 provides an overview of related work in this field. Section 5 discusses the importance of cloud computing for this project, while Section 6 presents the system architecture and design. Section 7 provides details on the algorithms and data structures used in the application, while Section 8 presents the testing and results. Section 9 discusses the deployment of the application on a public cloud platform, and finally, Section 10 presents the key conclusions and future directions of the project.

Twilio Programmable Video

Twilio Programmable Video is a cloud-based platform that allows developers to add real-time video and audio capabilities to their applications. With Programmable Video, developers can easily create video chat, conference, and live streaming applications that can be used across a variety of devices and platforms.

Programmable Video uses a REST API that allows developers to control every aspect of the video experience, from initiating and ending calls to controlling camera and microphone access. The API is easy to use and integrates seamlessly with other Twilio services, such as Twilio Authentication, Twilio Chat, and Twilio Sync.

At the core of Programmable Video is the Twilio Video SDK, which is available for a variety of platforms, including iOS, Android, JavaScript, and React Native. The SDK provides a set of easy-to-use tools that developers can use to build their own video applications. These tools include a pre-built UI for video calls, custom signaling and media servers, and advanced features like screen sharing and recording. One of the key advantages of Programmable Video is its scalability. The platform is designed to handle large numbers of concurrent users, with support for up to 50 participants in a single call. This makes it ideal for applications like virtual events, online classrooms, and remote work teams.

To ensure the security and privacy of users, Programmable Video includes a number of built-in security features. These include end-to-end encryption, secure tokens for user authentication, and secure network communication.

In conclusion, Twilio Programmable Video is a powerful platform that allows developers to easily add video and audio capabilities to their applications. With its easy-to-use API, pre-built SDKs, and advanced features, Programmable Video is an excellent choice for anyone looking to build real-time video applications that can scale to meet the needs of users around the world.

Twilio Functions

Twilio Functions is a serverless environment that allows developers to create and deploy custom code directly in the Twilio cloud. With Functions, developers can build custom logic for their Twilio applications without the need to set up and manage their own servers.

Functions is built on top of the AWS Lambda platform, which means that it can handle high levels of traffic and automatically scale to meet demand. It also integrates seamlessly with other Twilio services, such as Twilio Studio, Twilio API, and Twilio SMS.

One of the key advantages of Functions is its ease of use. Developers can create and deploy Functions directly from the Twilio Console, without the need to set up any infrastructure. Functions also supports multiple programming languages, including JavaScript, Python, and Ruby, which makes it accessible to a wide range of developers.

Functions are triggered by incoming requests from Twilio, such as an incoming SMS message or a phone call. When a request is received, Functions processes the request and returns a response. This allows developers to add custom logic to their Twilio applications, such as filtering incoming messages or processing user input.

Functions is also highly secure. It includes built-in support for HTTPS and TLS, and can be configured to restrict access to specific IP addresses or authentication credentials. This makes it an excellent choice for applications that require high levels of security, such as financial applications or healthcare applications.

Twilio Functions is a powerful serverless environment that allows developers to build custom logic for their Twilio applications without the need to manage their own servers. With its ease of use, support for multiple programming languages, and built-in security features, Functions is an excellent choice for anyone looking to add custom logic to their Twilio applications.

React

React is a popular JavaScript library that is widely used for building user interfaces. It provides developers with a set of tools for creating reusable UI components and managing the state of those components. When building a video conferencing application, React can be a great choice for the frontend, as it allows for easy management of complex user interfaces.

One of the key advantages of using React for a video conferencing application is its ability to create reusable components. This can be particularly useful for creating the various UI elements that make up a video conferencing application, such as the video feeds, chat boxes, and control panels. By creating these components once and reusing them throughout the application, developers can save time and reduce the risk of errors.

React also makes it easy to manage the state of the application. In a video conferencing application, there are many different states that need to be managed, such as the state of the video feeds, the state of the chat messages, and the state of the various controls. By using React's state management tools, developers can easily keep track of these states and ensure that the UI stays in sync with the underlying data.

Another advantage of React is its ability to handle complex UI interactions. In a video conferencing application, there are many different UI elements that need to interact with each other, such as the video feeds, the chat messages, and the various controls. By using React's event handling and component lifecycle methods, developers can easily manage these interactions and ensure that the UI remains responsive and intuitive.

React is a powerful tool for building the frontend of a video conferencing application. With its ability to create reusable components, manage complex UI interactions, and handle application state, React can help developers create a high-quality and user-friendly video conferencing application that meets the needs of users.

Javascript as backend

JavaScript is a popular programming language that is widely used for building web applications. Traditionally, JavaScript has been used primarily on the frontend of web applications, but with the advent of Node.js, it is now possible to use JavaScript as a backend language as well.

One of the key advantages of using JavaScript as a backend language is its ability to handle asynchronous I/O operations. This is particularly useful in web applications, where many requests are made to servers and databases, and responses need to be returned quickly.

JavaScript's event-driven architecture and support for callbacks and promises make it well-suited for handling these types of operations.

Another advantage of using JavaScript as a backend language is its popularity and wide adoption. Many developers are already familiar with JavaScript, so using it for both the frontend and backend of a web application can help reduce the learning curve and improve productivity. In addition, the large ecosystem of JavaScript libraries and frameworks, such as Express and Nest.js, make it easy to build complex and scalable backend systems.

JavaScript is also highly extensible and can be used for a wide range of backend tasks, such as server-side rendering, API development, and data processing. With the ability to run JavaScript on both the client and server side, developers can build highly interactive and responsive web applications that provide a seamless user experience.

In terms of performance, JavaScript as a backend language can be highly performant when used in conjunction with tools like Node.js and V8 engine. This allows for efficient and speedy execution of server-side code, making it a viable option for high-traffic and mission-critical web applications.

JavaScript is a versatile and powerful language that can be used as a backend language in web applications. Its ability to handle asynchronous I/O operations, popularity and wide adoption, extensibility, and performance make it a compelling choice for developers looking to build web applications that require both frontend and backend functionality.

twilio programmable video room types

Twilio Programmable Video is a cloud-based video communication platform that allows developers to add video, audio, and chat capabilities to their applications.

One of the key features of Twilio Programmable Video is its support for different types of rooms, which are virtual spaces where participants can communicate with each other.

There are three types of rooms available in Twilio Programmable Video: peer-to-peer rooms, group rooms, and composite rooms:

1. **Peer-to-Peer Rooms:** Peer-to-peer rooms are rooms where two participants can communicate directly with each other, without the need for a server to manage the connection. This type of room is ideal for one-on-one conversations, such as video calls between friends or colleagues. Peer-to-peer rooms are easy to set up and use, and they provide a low-latency communication experience.
2. **Group Rooms:** Group rooms are rooms where multiple participants can communicate with each other, using a server to manage the connection. This type of room is ideal for small group conversations, such as team meetings or virtual classrooms. Group rooms provide features such as video layout management, enabling the layout of the video feeds to be customized, and network resiliency, which ensures that the connection remains stable even in low-bandwidth situations.
3. **Composite Rooms:** Composite rooms are rooms that combine both peer-to-peer and group room functionality. This type of room allows participants to communicate directly with each other, as well as with other participants in the room. This is useful in situations where there are multiple sub-groups within a larger group, and those sub-groups need to communicate with each other as well as the larger group. Composite rooms provide features such as the ability to control which participants can communicate with each other, and the ability to switch between peer-to-peer and group communication modes.

In conclusion, Twilio Programmable Video provides three types of rooms to support different types of

communication scenarios: peer-to-peer rooms for one-on-one conversations, group rooms for small group conversations, and composite rooms for situations where there are multiple sub-groups within a larger group. Each room type provides specific features and functionality to enable developers to build powerful and flexible video communication applications.

How go and p2p rooms are working

Twilio Programmable Video provides developers with the ability to create P2P rooms, where two participants can communicate directly with each other without the need for a server to manage the connection.

In a P2P room, Twilio acts as a signaling server, which facilitates the initial connection between the two participants, after which the participants communicate directly with each other.

When a participant joins a P2P room, Twilio assigns them a unique identifier called a token, which is used to authenticate them and grant them access to the room. The participant then sends their token to the other participant, either directly or through a signaling mechanism such as WebSockets or a third-party service. Once the other participant receives the token, they can use it to connect to the room and establish a direct peer-to-peer connection.

Once the two participants are connected, they can send audio, video, and data directly to each other without the need for a server to relay the communication. This provides a low-latency and high-quality communication experience, as the communication is not affected by network delays or server load. P2P rooms are particularly useful for applications that require low-latency communication, such as online gaming or one-on-one video calls. However, P2P rooms have some limitations, such as a lack of network resiliency in low-bandwidth situations, which can result in dropped connections or poor video quality. In addition, P2P rooms are not suitable for group conversations, as each participant would need to establish a direct connection with every other participant in the room, which can be resource-intensive and impractical.

P2P rooms in Twilio Programmable Video allow two participants to communicate directly with each other without the need for a server to manage the connection, providing a low-latency and high-quality communication experience. However, P2P rooms have some limitations and are not suitable for group conversations or situations where network resiliency is required.

Two solutions for token service- twilio serverless function and our own javascript server

1. **Twilio Serverless Functions:** Twilio provides a serverless function environment where developers can create and deploy small pieces of code that run in the cloud. Using Twilio's serverless functions, developers can create a token service that generates and returns access tokens for Programmable Video. These tokens can be customized to control access to specific rooms, grant certain permissions to participants, and set an expiration time for the token.

To use Twilio's serverless functions for token service, developers need to write JavaScript code that generates access tokens for Programmable Video. The code can be deployed to Twilio's serverless environment using the Twilio CLI, and can be triggered by HTTP requests. Twilio's serverless functions provide a scalable and reliable solution for token generation, as the code is executed in the cloud and can handle high volumes of traffic.

2. **Custom JavaScript Server:** Alternatively, developers can implement their own JavaScript server to generate access tokens for Programmable Video. This approach requires developers to have a good understanding of server-side JavaScript programming and security best practices. Developers need to implement an API that generates access tokens based on a set of rules, such as the room name, participant identity, and expiration time.

Implementing a custom JavaScript server for token service provides more flexibility and control over the token generation process, as developers can customize the logic and rules used to generate the tokens. However, this approach also requires developers to maintain and secure their own server infrastructure, which can be a significant overhead.

There are several architectural styles that can be used to create a video calling or conferencing app, depending on the specific requirements and constraints of the application.

Here are some of the most common architectural styles for video calling or conferencing apps:

1. **Client-Server Architecture:** The client-server architecture is a common architectural style for web-based video calling or conferencing apps. In this architecture, the client application runs on the user's device and communicates with a server over the internet to exchange video and audio data. The server acts as a bridge between the participants, relaying data between them in real-time. This architecture can be further divided into two types:

- **Centralized server architecture:** In this architecture, a central server manages the video and audio streams, and each participant in the conference connects to this server. The server is responsible for processing and relaying the video and audio data to all participants in the conference. This approach is commonly used for large-scale video conferencing apps like Zoom, Google Meet, and Microsoft Teams.

- **Decentralized server architecture:** In this architecture, each participant in the conference connects to a peer-to-peer network and exchanges data directly with other participants. This approach can be used for smaller-scale video calling apps or conferences with a limited number of participants.

2. **Peer-to-Peer Architecture:** The peer-to-peer (P2P) architecture is another common architectural style for video calling or conferencing apps. In this architecture, the participants in the conference connect directly to each other, without the need for a central server. The participants exchange data directly in real-time, resulting in low latency and high-quality communication. This architecture is commonly used for one-on-one video calling apps or small-scale conferences.

3. **Hybrid Architecture:** The hybrid architecture combines the client-server and peer-to-peer architectures to create a more scalable and resilient system. In this architecture, the participants in the conference connect to a central server, which acts as a signaling and control server. Once the participants are connected, they exchange data directly with each other in a peer-to-peer network, without the need for the server to relay the data. This approach provides the low latency and high-quality communication of a P2P architecture, while still providing the scalability and resilience of a client-server architecture. In summary, there are several architectural styles that can be used to create a video calling or conferencing app, including client-server architecture, peer-to-peer architecture, and hybrid architecture. The choice of architecture depends on the specific requirements of the application, such as the number of participants, the need for scalability and resilience, and the desired communication quality.

Access token from twilio

In Twilio, access tokens are used to authenticate a user's identity and grant them access to Twilio Programmable Video Rooms. Access tokens are time-limited and provide temporary access to a specific resource, in this case, the Twilio Programmable Video Rooms.

To generate an access token in Twilio, you need to create a server-side implementation in one of the supported languages such as Node.js, Python, Ruby, or PHP. This implementation should use the Twilio API and generate the access token on demand, whenever a user needs to join a video room.

Here are the general steps to generate an access token using the Twilio API:

1. First, you need to create an account on Twilio and obtain the Account SID and Auth Token, which are used to authenticate your application to the Twilio API.
2. Next, you need to create a Twilio API key and secret, which are used to sign and verify access tokens. You can create API keys and secrets in the Twilio Console.
3. Once you have the necessary credentials, you can use the Twilio API to generate an access token. The access token contains a unique identity for the user and a set of permissions that determine what actions the user can perform in the video room, such as publishing or subscribing to video and audio streams.
4. To generate an access token, you need to construct a JSON Web Token (JWT) payload that contains the user's identity and the desired permissions. You then sign the JWT payload using your Twilio API key and secret to create a secure access token.
5. Finally, you can pass the access token to the user's device, which can use it to authenticate and connect to the Twilio Programmable Video Rooms.

twilio cli

The Twilio CLI is a command-line interface tool provided by Twilio to help developers interact with the Twilio APIs and services from the command line. With the Twilio CLI, you can manage and configure your Twilio resources such as phone numbers, messaging services, and programmable video rooms, as well as create and deploy Twilio Serverless Functions and Twilio Studio Flows.

Here are some of the key features of the Twilio CLI:

1. Resource management: You can manage your Twilio resources such as phone numbers, messaging services, and programmable video rooms, directly from the command line using simple commands.
2. Function deployment: You can create, deploy and manage Twilio Serverless Functions from the command line, making it easy to quickly test and iterate on your code.
3. Studio Flow management: You can create and manage Twilio Studio Flows from the command line, allowing you to automate and customize your communications flows.
4. Integration with other tools: The Twilio CLI integrates with popular development tools such as Git, enabling you to version control your Twilio configurations and deployments.
5. Contextual help: The Twilio CLI provides contextual help for every command, making it easy to quickly understand what a command does and how to use it.
6. Scriptable: The Twilio CLI can be easily integrated into your development workflows and build processes, allowing you to automate tasks and streamline your development processes.

In summary, the Twilio CLI is a powerful tool that allows developers to manage and interact with Twilio APIs and services from the command line. It provides a convenient way to manage and configure Twilio resources, deploy serverless functions, and automate communications flows. The Twilio CLI is a valuable tool for any developer working with Twilio services and APIs.

twilio serverless plugin

The Twilio Serverless Plugin is a tool that extends the functionality of the Twilio CLI to enable developers to create and deploy serverless functions to the Twilio Runtime environment directly from the command line. This plugin makes it easier for developers to build, deploy, and manage serverless applications that are integrated with Twilio services and APIs.

Here are some of the key features of the Twilio Serverless Plugin:

1. Easy deployment: With the Twilio Serverless Plugin, developers can quickly create and deploy serverless functions to the Twilio Runtime environment using simple CLI commands.
2. Integrated with Twilio services: The Twilio Serverless Plugin is designed to work seamlessly with Twilio services and APIs, allowing developers to build serverless applications that are tightly integrated with Twilio.
3. Multiple language support: The Twilio Serverless Plugin supports multiple programming languages, including JavaScript, Python, and Java, making it easy for developers to build serverless applications using their preferred language.
4. Environment management: The Twilio Serverless Plugin includes tools for managing different environments, such as development, staging, and production, making it easy to deploy and manage serverless applications in different environments.
5. Version control: The Twilio Serverless Plugin includes built-in version control capabilities, allowing developers to track changes to their serverless functions and roll back to previous versions if necessary.
6. Test locally: The Twilio Serverless Plugin includes a built-in testing framework that enables developers to test their serverless functions locally before deploying them to the Twilio Runtime environment.

redux

Redux is a state management library for JavaScript applications, particularly those built with React. It provides a centralized store for managing application state and a set of rules for updating that state in response to user actions or other events. Redux works by having a single store that holds the entire application state, and any changes to that state are made through dispatching actions.

What is routing and adding https

Routing refers to the process of mapping URLs to different parts of an application or website. In other words, it determines how the application responds to different requests based on the URL or path that is requested. Routing is commonly used in web applications to provide a clear and structured way to navigate between different pages or sections of the application.

In a typical web application, routing is usually handled by a routing library or framework, such as React Router, Express Router, or Vue Router. These libraries allow you to define different routes and how the application should respond to each of them.

Adding HTTPS to a web application involves securing the connection between the client (e.g. a web browser) and the server by encrypting the data that is transmitted. This is important for protecting sensitive information, such as login credentials or payment information, from being intercepted or tampered with.

To add HTTPS to a web application, you need to obtain an SSL/TLS certificate from a trusted certificate authority (CA) and install it on your web server. Once the certificate is installed, you can configure your web server to use HTTPS by redirecting all HTTP traffic to HTTPS and specifying the appropriate SSL/TLS settings.

In addition to obtaining and installing an SSL/TLS certificate, there are several other best practices for securing a web application. These include using secure coding practices, enforcing strong password policies, implementing multi-factor authentication, and regularly updating software and security patches.

twilio room statistics

Twilio Programmable Video provides a variety of statistics and metrics that you can use to monitor the quality and performance of your video calls and conferences. These statistics include both participant-level and room-level metrics, such as audio and video quality, network conditions, and device information.

To access room statistics, you can use the Twilio REST API or the Twilio SDKs to query the Room object for a given room. The Room object contains a `room_status` property that indicates the current status of the room, as well as a `room_type` property that specifies the type of room (e.g. peer-to-peer or group room).

Once you have obtained the Room object, you can use the `getRoomStats()` method to retrieve the current statistics for the room. This method returns an array of `ParticipantStats` and `RemoteParticipantStats` objects that contain information about each participant in the room, as well as aggregate statistics for the room as a whole.

Some of the metrics that you can access through the Room object include:

- Audio levels: The current audio levels for each participant, including average and maximum levels.
- Video quality: The current resolution, frame rate, and bitrate for each video track in the room.
- Network conditions: The current network quality and round-trip time (RTT) for each participant.
- Device information: The type and version of device being used by each participant.
- Connection information: The current connection state and status code for each participant.

By monitoring these metrics and using them to analyze the performance and quality of your video calls and conferences, you can identify and address issues that may be affecting the user experience. For example, if you notice that certain participants are experiencing poor network conditions, you may be able to improve the quality of the call by adjusting the network settings or suggesting that participants switch to a different network or device.

Features:

1. Video and audio calling: The application can support both one-on-one and group video and audio calls, allowing users to communicate in real-time with one or more participants.
2. Screen sharing: Users can share their screens with other participants in the call, enabling collaboration and presentation sharing.
3. Chatting: Users can send text messages to each other during the call, allowing them to communicate silently without interrupting the conversation.
4. Recording: The application can allow the recording of the call for later playback or sharing with others.
5. Scheduling: Users can schedule video conferencing meetings in advance, with the application sending out invites and reminders to participants.
6. User authentication: The application can authenticate users to ensure that only authorized users can access the video conferencing functionality.
7. Real-time notifications: Users can receive real-time notifications when a participant joins or leaves the call, or when a new message is received in the chat.
8. Participant management: The host or moderator of the call can manage the participants in the call, such as muting or unmuting participants, removing participants, and promoting or demoting participants to co-hosts.
9. Customizable layout: The application can offer customizable layouts for the video conferencing interface, such as showing a gallery view or a speaker view, or allowing users to pin certain participants' videos.
10. Quality of Service (QoS): The application can measure and optimize network quality to ensure high-quality video and audio calls, including optimizing bandwidth usage, adjusting video quality, and reducing latency.

WebRTC

WebRTC is a free, open-source project that enables real-time communication between web browsers and mobile applications via simple APIs. It is a powerful tool that enables developers to create high-quality video, audio, and data communications between browsers and other devices without the need for plugins or third-party software.

WebRTC uses several standard protocols, such as STUN (Session Traversal Utilities for NAT) and ICE (Interactive Connectivity Establishment), to enable direct communication between devices. WebRTC also supports peer-to-peer (P2P) communication, which means that data is sent directly between devices, rather than through a central server.

WebRTC's main components include:

- getUserMedia API: This API allows the user to access the camera and microphone of their device, which is necessary for capturing audio and video.
- RTCPeerConnection API: This API establishes a connection between two devices, allowing them to exchange audio, video, and data.
- RTCDataChannel API: This API enables two devices to exchange arbitrary data in real-time, even when a P2P connection is not possible.

WebRTC is used in a variety of applications, including video conferencing, file sharing, online gaming, and more. It offers several benefits, including:

- Easy integration: WebRTC can be easily integrated into web applications using simple APIs, making it an accessible and straightforward tool for developers.
- High-quality audio and video: WebRTC supports high-quality audio and video, making it an excellent tool for video conferencing and other real-time communication applications.
- Security: WebRTC uses encryption to ensure that communication between devices is secure and private.
- Low latency: WebRTC uses P2P communication, which reduces latency and improves overall performance.

Twilio Programmable Video leverages WebRTC technology to enable real-time video and audio communication between participants in a video call. WebRTC provides the underlying technology for Twilio Programmable Video to establish peer-to-peer (P2P) connections between participants, allowing them to transmit audio and video directly to one another without the need for a central server. However, Twilio Programmable Video also provides additional features and capabilities beyond what

WebRTC alone can offer, such as:

1. **Server-side recording:** Twilio Programmable Video can record video and audio on the server-side, providing a reliable and secure way to record calls.
2. **Scalability:** Twilio Programmable Video can scale to support thousands of participants in a single call, without sacrificing performance or quality.
3. **Customization:** Twilio Programmable Video provides APIs and tools to customize the look and feel of the video call interface and add additional functionality, such as chat or screen sharing.
4. **SDKs:** Twilio Programmable Video provides SDKs for a variety of platforms, including iOS, Android, and web, making it easy for developers to integrate video calling functionality into their applications.

Signaling Server

A signaling server is a server that enables real-time communication between two or more devices. It is responsible for facilitating the exchange of information between devices, allowing them to establish a connection and communicate with one another.

In the context of WebRTC, a signaling server is used to exchange signaling messages between devices, such as SDP (Session Description Protocol) messages and ICE (Interactive Connectivity Establishment) candidates. These signaling messages contain information about the devices, such as their IP addresses, ports, and codecs, and are used to negotiate the parameters of the communication session.

The signaling server does not transmit the actual audio or video data; instead, it simply enables the devices to establish a peer-to-peer (P2P) connection so that they can transmit data directly between each other. Once the connection is established, the devices can transmit data without the need for the signaling server.

There are several different types of signaling servers, including:

1. **WebSockets:** WebSocket is a protocol that provides a full-duplex communication channel over a single TCP connection. It is often used as a signaling server for WebRTC applications.
2. **HTTP:** HTTP is a protocol that is commonly used for signaling in WebRTC applications. It uses HTTP requests and responses to exchange signaling messages between devices.
3. **SIP:** SIP (Session Initiation Protocol) is a signaling protocol that is often used in traditional telephony systems. It can also be used as a signaling server for WebRTC applications.

STUN server

A STUN (Session Traversal Utilities for NAT) server is a type of server used in WebRTC to help establish a direct connection between two devices across the internet. STUN servers are used when one or both devices are behind a NAT (Network Address Translation) router, which can prevent direct communication between the devices.

When a device behind a NAT router wants to communicate with another device, it needs to first determine its public IP address and port number. This information is needed so that the other device can send packets directly to the first device. However, the private IP address and port number of the device are not visible to the other device, as they are translated by the NAT router.

This is where a STUN server comes in. The STUN server helps the device behind the NAT router determine its public IP address and port number by sending it a STUN request. The device responds to the request, and the STUN server sends back the IP address and port number that the device is using to communicate with the internet.

Once the device has this information, it can use it to establish a direct connection with the other device, without going through a relay server. This direct connection is faster and more reliable than going through a relay server, which can be important for real-time communication applications like WebRTC. STUN servers are commonly used in WebRTC applications, and many STUN servers are publicly available for use. It is also possible to run your own STUN server if you need more control over the process.

TURN server

A TURN (Traversal Using Relays around NAT) server is a type of server used in WebRTC to help establish a direct connection between two devices across the internet when a direct peer-to-peer connection is not possible. TURN servers are used when both devices are behind restrictive firewalls, or if a NAT router does not support NAT traversal techniques like STUN.

When a direct peer-to-peer connection cannot be established, the WebRTC application can fall back to using a TURN server to relay the audio and video streams between the two devices. The TURN server acts as a relay between the two devices, receiving data from one device and forwarding it to the other. However, using a TURN server comes with some drawbacks. Firstly, it adds extra latency to the connection, as data must travel through the TURN server. Secondly, using a TURN server can be expensive, as it requires additional bandwidth and processing power to relay the audio and video streams.

TURN servers are commonly used in WebRTC applications as a fallback option when direct peer-to-peer connections are not possible. Many TURN servers are publicly available for use, but it is also possible to run your own TURN server if you need more control over the process.

It is worth noting that in some cases, both STUN and TURN servers may be required to establish a successful WebRTC connection. STUN servers can help devices behind NAT routers determine their public IP addresses and ports, while TURN servers can be used as a fallback option when direct peer-to-peer connections are not possible.

SDP

SDP stands for Session Description Protocol, which is a format used to describe multimedia sessions for communication over IP networks. In the context of WebRTC, SDP is used to negotiate and exchange information between two devices to establish a connection for real-time audio and video communication. When two devices want to establish a WebRTC connection, they exchange SDP messages to negotiate the parameters of the connection. The SDP messages contain information about the audio and video codecs that each device supports, the IP addresses and ports that each device is using, and other session-specific details.

The SDP messages are exchanged using a signaling server, which is responsible for relaying the messages between the two devices. The signaling server does not handle the actual audio and video streams, but rather facilitates the negotiation of the connection between the two devices.

Once the SDP messages have been exchanged and the connection parameters have been agreed upon, the devices can establish a direct peer-to-peer connection for the audio and video streams. The SDP messages are only used during the negotiation phase of the connection, and are not used for the actual audio and video communication.

SDP is an important part of the WebRTC protocol, as it allows devices to negotiate the parameters of a connection and establish a direct peer-to-peer connection for real-time audio and video communication.

ICE candidates

In WebRTC, ICE (Interactive Connectivity Establishment) candidates are used to help establish a direct peer-to-peer connection between two devices. When two devices want to establish a WebRTC connection, they exchange ICE candidates to discover the best path for data to travel between them.

An ICE candidate represents a possible network endpoint for a WebRTC connection. The candidate includes information such as the device's IP address, port number, and transport protocol (UDP or TCP).

Devices exchange ICE candidates to discover each other's IP addresses and ports, and to identify the best candidate for establishing a direct peer-to-peer connection.

The ICE process begins with each device gathering a list of its own ICE candidates, based on its available network interfaces and transport protocols. The device then sends its list of candidates to the other device, and receives a list of candidates in return. The two devices then compare their lists of candidates to determine the best path for data to travel between them.

If a direct peer-to-peer connection is not possible (for example, if both devices are behind restrictive firewalls), the WebRTC connection can fall back to using a TURN server to relay the audio and video streams between the devices.

How direct connection is established between users

In WebRTC, direct connections between users are established using a combination of ICE (Interactive Connectivity Establishment) and SDP (Session Description Protocol) protocols. Here's a high-level overview of the process:

1. Each user's device gathers a list of its own ICE candidates, based on its available network interfaces and transport protocols. ICE candidates include information such as the device's IP address, port number, and transport protocol (UDP or TCP).
2. The device sends its list of ICE candidates to the other user's device, and receives a list of ICE candidates in return.
3. The two devices use SDP messages to negotiate and agree on the best ICE candidate for establishing a direct peer-to-peer connection.
4. If a direct peer-to-peer connection is established, the two devices can exchange real-time audio and video data directly, without the need for a server to relay the data.
5. If a direct peer-to-peer connection cannot be established (for example, if both devices are behind restrictive firewalls), the WebRTC connection can fall back to using a TURN server to relay the audio and video streams between the devices.

Once a direct peer-to-peer connection has been established between the two users, they can exchange real-time audio and video data directly, without the need for a server to relay the data. This allows for faster and more efficient communication, as the data does not have to be transmitted through a third-party server.

Git

Git is a distributed version control system that allows developers to track changes to code over time and collaborate with others on software projects. It was developed by Linus Torvalds in 2005 and has since become one of the most popular version control systems used in the software development industry. GitHub is a web-based hosting service for Git repositories. It provides a platform for developers to store their code in a centralized location, collaborate with others on projects, and track issues and bugs. GitHub also offers a range of features such as pull requests, code reviews, and integrations with other tools like CI/CD pipelines.

Here are some common Git commands:

1. `git init` - initializes a new Git repository in the current directory.
2. `git add <file>` - stages a file for commit.
3. `git commit -m "commit message"` - creates a new commit with the staged changes and a commit message.
4. `git status` - shows the status of the repository, including any changes that have been made and staged.
5. `git log` - shows the commit history of the repository.
6. `git branch` - lists all branches in the repository.
7. `git checkout <branch>` - switches to the specified branch.
8. `git merge <branch>` - merges the specified branch into the current branch.
9. `git push` - pushes the local commits to the remote repository.
10. `git pull` - pulls the latest changes from the remote repository.

HTML, CSS, JavaScript

HTML, CSS, and JavaScript are three of the core technologies used to create websites and web applications. Here is a brief overview of each:

1. **HTML (Hypertext Markup Language)** - HTML is the standard markup language used to create the structure and content of web pages. It defines the elements and attributes that make up a webpage, such as headings, paragraphs, images, links, and more.
2. **CSS (Cascading Style Sheets)** - CSS is a stylesheet language used to define the presentation and layout of web pages. It is used to control the styling of HTML elements, including colors, fonts, margins, padding, and more.
3. **JavaScript** - JavaScript is a programming language used to add interactivity and dynamic behavior to web pages. It can be used to create animations, handle user input, and interact with server-side APIs.

Together, HTML, CSS, and JavaScript form the foundation of most web pages and web applications. They are typically used in conjunction with server-side programming languages, such as PHP or Python, and databases, such as MySQL or MongoDB, to create dynamic and interactive websites. By mastering these technologies, developers can create beautiful and functional web applications that run in a browser.

PROBLEM STATEMENT

Problem statement for this project is to develop a video-conferencing web application using Twilio API and deploy it on a cloud platform. With the rise of remote work and online communication, there is an increasing demand for video conferencing solutions that are easy to use and accessible from anywhere.

However, building a video conferencing application from scratch can be a complex and time-consuming task. Additionally, deploying the application on a cloud platform can be challenging due to the need for scalability and reliability.

Therefore, the goal of this project is to create a video-conferencing web application that can be easily deployed on a cloud platform, enabling users to communicate with each other through video and audio calls. The application should be user-friendly, secure, and capable of handling multiple simultaneous connections without compromising the quality of the call.

The deployment on a cloud platform should provide the necessary scalability and reliability to ensure that the application can handle a large number of users without downtime or performance issues.

OBJECTIVE

The objective of this project is to create a web-based video conferencing application using Twilio that can be used by individuals or businesses for remote communication.

The application should be easy to use and offer reliable video and audio quality for smooth communication. The project aims to provide a cost-effective and accessible solution to enable people to connect with each other remotely.

Additionally, the objective is to deploy the application on a public cloud platform to make it easily accessible to users from anywhere in the world.

The project also aims to leverage cloud technology to ensure scalability, flexibility, and high availability of the application.

- To develop a user-friendly and reliable video conferencing web application using the Twilio API that allows for seamless communication between individuals and groups.
- To provide a platform that enables users to make voice and video calls, share screens, and send text messages while maintaining high-quality audio and video.
- To design the application in a way that prioritizes security, protecting user information and preventing unauthorized access.
- To ensure that the application is user-friendly, with an intuitive interface and easy-to-use controls that require minimal technical knowledge.

RELATED WORK

Some of the notable related works are as follows:

1. **Zoom:** Zoom is a popular video conferencing software that offers cloud-based solutions for businesses and individuals. It has gained immense popularity during the COVID-19 pandemic due to its ease of use, reliability, and a wide range of features. Zoom allows users to conduct online meetings, webinars, and video conferences with up to 1000 participants.
2. **Google Meet:** Google Meet is a video conferencing software that is integrated with the G Suite platform. It allows users to conduct online meetings, webinars, and video conferences with up to 250 participants. Google Meet is cloud-based and can be easily accessed from any device.
3. **Microsoft Teams:** Microsoft Teams is a chat-based collaboration platform that offers video conferencing and other communication tools. It is integrated with the Microsoft Office Suite and can be easily accessed from any device. Microsoft Teams allows users to conduct online meetings, webinars, and video conferences with up to 10,000 participants.
4. **Cisco Webex:** Cisco Webex is a video conferencing software that offers cloud-based solutions for businesses and individuals. It allows users to conduct online meetings, webinars, and video conferences with up to 1000 participants. Cisco Webex is known for its reliability and security features.
5. **Jitsi Meet:** Jitsi Meet is an open-source video conferencing software that can be easily deployed on the cloud. It is known for its ease of use, reliability, and security features. Jitsi Meet allows users to conduct online meetings, webinars, and video conferences with up to 75 participants.

There are several video conferencing apps in the market that offer similar functionalities to the one you have developed using Twilio. Some of the popular ones include:

1. **Zoom** - Zoom is one of the most popular video conferencing apps in the market, offering features like screen sharing, recording, and virtual backgrounds. It uses a combination of WebRTC, RTMP, and HLS to stream video and audio, and also offers support for cloud recording and live streaming.
2. **Microsoft Teams** - Microsoft Teams is a collaboration platform that includes video conferencing, messaging, and file sharing. It uses the WebRTC protocol for video and audio transmission and offers features like screen sharing, recording, and background blur.
3. **Google Meet** - Google Meet is a video conferencing app that allows users to join meetings from their web browser or mobile device. It uses WebRTC for video and audio transmission and offers features like screen sharing, recording, and real-time captions.

Following are some literature reviews that suggest why cloud is important and required in the growing demand of video conferencing app:

1. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4432776/>

This review compares traditional videoconferencing systems with cloud-based videoconferencing systems. It highlights the differences between the two and explains the methods used to evaluate them. The results suggest that the technology is evolving rapidly and hands-on experience is necessary to fully assess the relevance of cloud conferencing technology. The review provides a template for evaluating the appropriateness of systems. It distinguishes cloud videoconferencing from traditional videoconferencing by explaining the key difference of client conferencing software accessing videoconferencing software on servers in the cloud. Traditional videoconferencing is done through installed appliances in a room and uses the H.323

communication standard for interoperability. The review also mentions terms related to cloud conferencing such as WebRTC, unified communication, and video as a service. The purpose of the review is to provide a framework for assessing cloud videoconferencing in relation to local needs and to determine if migration from traditional to cloud systems is justified. The findings show that cloud videoconferencing systems are characterized by several key features, including video and audio encoding, multipoint conferencing, compatibility with different operating systems and computing platforms, interoperability, security, content sharing, user-friendly interfaces, archiving capabilities, and Webcasting options. The results of the review suggest that cloud-based videoconferencing systems offer several advantages over traditional systems, including increased flexibility, scalability, and ease of use.

ATTRIBUTES	TRADITIONAL SYSTEMS	CLOUD SYSTEMS
Video quality	Built-in video resolution	Potential windowing/resolution issues
Audio quality	Built-in audio quality control	Potential echo/feedback and loudness issues
Multipoint scalability	Difficult, may need special devices	Built-in multipoint capability, theoretically unlimited
OS/hardware	Built-in OS and PTZ camera control	Need to appraise computer capabilities compatibility, work around limited camera control
Interoperability	High, follows H.323 standard	Low, not standardized
Content sharing	Built-in but limited	Built-in with multiple sharing features
Security/network management	Network management/coordination required	Little network management needed, none for some systems
Interface	Difficult, assumes trained users	Easy, intuitive graphical interfaces intended for anyone
Archiving and Webcasting	Limited	Built-in/easy for some systems
Cost	Expensive, especially for end-point appliances	Expensive for servers, cheaper for clients

Source: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4432776/>

2. <https://www.frontiersin.org/articles/10.3389/feduc.2021.752522/full>

The COVID-19 pandemic has led to a decrease in physical interaction and an increase in the adoption of e-conferencing for conducting business globally. The virtualization of meetings promotes collaboration and helps meet the goals of organizations. Web conferencing has proven to be beneficial in the education sector, allowing higher learning institutions to expand access to instructors online while also creating new learning experiences. The study describes the use of web conferencing in education, its adoption and influence on teaching and learning, the benefits and disadvantages, and provides examples of two web-conferencing platforms (Zoom and Google Meet). The future of web-conferencing in education is also discussed. Web conferencing is a tool for virtual meetings, presentations, and collaborations that allows for real-time video content to be shared with a large group through the Internet. In Thailand, the government passed regulations in 2014 allowing for secure online meetings through electronic devices. Thailand has embraced online learning and has personalized, facilitative, and responsive education through technology. The adoption of information and technological systems, like web conferencing, benefit the education field by allowing for easy access to data and

improving collaboration. Collaboration is important in education and is essential for the success of an organization. Investment in collaboration technology and a supportive culture is necessary to realize meaningful collaboration. There are various tools, like e-mail, audio, and video conferencing, available to aid in the collaborative learning process.

3. <https://iopscience.iop.org/article/10.1088/1742-6596/1679/3/032078/pdf>

The widespread use of technology has led to the popularization of cloud computing and cloud services. Distributed cloud video conferencing technology provides easily scalable computing resources and allows for the delivery of infrastructures, platforms, and applications as services, benefiting many users including businesses. Cloud computing reduces the total cost of ownership of IT infrastructure, increases its fault tolerance, allows access to data from almost any device through the internet, and has high elasticity. One of the most popular cloud services is video conferencing, which has become increasingly important as remote work and remote collaboration have become more prevalent. Currently, there are several companies offering video conferencing services such as Cisco, Zoom, Microsoft, and LogMeIn.

Video conferencing technology can be classified into three types: hardware, software, and cloud. Cloud video conferencing is currently the most popular type. The quality of video communication is an important aspect of video conferencing and users expect improvements in this area, as well as increased integration with other information systems, applications, and services. The increasing demand for video conferencing and the limitations of existing services make it necessary to develop a new, distributed cloud video conferencing service.

The authors of the paper have developed a distributed cloud video conferencing service for use in the higher education system. The service's architecture is based on 5 types of servers: an application server, a database server, a mail server, a web socket server and SDN control servers. The network infrastructure is built on SDN technology, allowing for effective interaction between servers and management of video data streams. The proposed approach has been found to be effective in higher education and related fields.

4. [Measurement Study of Multi-party Video Conferencing Yue Lu, Yong Zhao, Fernando Kuipers, and Piet Van Mieghem](#)

This paper explores the rise in telecommuting and video conferencing due to COVID-19 restrictions on mass gatherings and social distancing, as well as the potential cyber security risks posed by these technologies. The paper provides a comprehensive overview of the vulnerabilities, threats, and impacts of cyber-attacks related to telecommuting and video conferencing. The paper is divided into five sections: Introduction, Components of telecommuting and video conferencing applications, Challenges of remote work and their cyber security implications, Business and technical impacts of WFH-based cyber-attacks. The shift to WFH has brought about several challenges from a technological, cybersecurity, and philosophical perspective. Technologically, there are several challenges to consider, including employee's role in the organization, their access to company assets, psychological balance to work alone, IT security awareness, the reliability of technology, and the convenience of the remote working environment. Geography also plays a role, with some regions having access to reliable internet services while others face issues with power supply and bandwidth. Additionally, working from home could also lead to data errors from distractions like household chores and personal emailing. Cybersecurity risks include insecure networks, poor network conditions, and inadequate bandwidth, which could lead to cyber attacks on remote communication systems. The paper is about the investigation and analysis of the characteristics and quality of experience of four representative multi-party video conferencing applications. The aim of the study is to answer questions about how these applications work, what

resources they need, what the Quality of Experience (QoE) is, what the bottleneck is in providing multi-party video conferencing over the internet, and what technology and architecture offer the best QoE. The study surveyed 18 popular video conferencing applications and chose four representative ones to measure, including Mebeam, Qnext, Vsee, and Nefsis.

5. <https://dl.acm.org/doi/pdf/10.1145/3487552.3487842>

This passage describes the results of a study on three modern video conferencing applications (VCAs): Zoom, Google Meet, and Microsoft Teams. The study aimed to understand the resource requirements and performance of these VCAs under different network conditions. The study found that the average utilization of an unconstrained link varied between 0.8 Mbps and 1.9 Mbps, and some VCAs required up to 50 seconds to recover to steady state after temporary reductions in capacity. The study also found differences in proprietary congestion control algorithms that result in unfair bandwidth allocations. For example, one Zoom video conference can consume more than 75% of the available bandwidth when competing with another VCA. The study also found that the utilization of some VCAs decreased as the number of participants increased, and that the viewing mode of one participant can affect the upstream utilization of other participants. This paper is motivated by the increasing reliance on video conferencing applications (VCAs) due to the COVID-19 pandemic, which has highlighted disparities in internet access for remote education and work. The authors aim to answer questions about the network requirements for VCAs, including their network utilization, performance under different link capacities, response to disruptions, competition with other applications, and the effect of usage modalities on network utilization. The authors perform controlled experiments with emulated network conditions and collect data using APIs. Their findings have implications for network management and policy, including questions about the throughput needed to support quality video conferencing.

6. [Central European Journal of Educational Research 2\(2\) 2020. 84–92.](#)

Germann et al. analyzed the possibility of remote teaching during a pandemic, and Faherty et al. explored the preparedness of schools for such a situation. Online learning rearranges the world of homework and exams, but it also poses challenges such as the need for new technologies to avoid cheating and the possibility of a student lacking access to the tools necessary for online education. Social disparities increase with distance education and students without access to education face serious long-term problems, including a reduced chance of meeting year-end requirements, increased chances of dropping out, and permanent disadvantage.

Video conferencing allows real-time transmission and reception of audio and video data over a network between users who are at a distance from each other. Skype and Zoom are popular videoconferencing tools used for teleconferencing, telecommuting, distance learning, and social networking. Skype is free for users for calls inside the app and can be used for instant messaging, voice and video communication, and conference calls. Zoom, on the other hand, has a user-friendly interface and offers face-to-face chat, group video conferencing, screen sharing, recording appointments, etc. Zoom's use is free for up to 100 participants and has a time limit of 40 minutes for calls with more than two people. Paid subscriptions are available for longer or larger conferences. Zoom experienced a huge increase in users during the COVID-19 pandemic due to the exponential growth of teleworking and distance learning.

7. <https://doisrpska.nub.rs/index.php/JTTTP/article/view/6562/6430>

Video conferencing has become an essential tool for various fields including business, education, health, and others. It provides a way for students to communicate with each other and teachers through synchronous two-way audio and video over the internet. This type of technology expands traditional teaching methods by allowing the use of digital images and videos. Video conferencing enables distance education and can reach new target groups, making education more flexible and accessible. The use of video conferencing has been facilitated by the growth of internet services and is considered a critical component of e-learning technology.

WHY DO WE NEED CLOUD FOR THE PROJECT

Cloud computing has become increasingly popular in recent years, as it offers a number of benefits over traditional on-premise infrastructure. In the context of your video conferencing application, cloud computing provides a number of advantages that can help improve the performance, scalability, and reliability of your application.

One of the key benefits of cloud computing is its ability to scale dynamically to meet changing demands. With cloud computing, you can easily provision additional resources such as compute power, storage, and networking as your application's needs grow. This is particularly important for video conferencing applications, as the amount of traffic and resource requirements can vary greatly depending on the number of participants, quality of the video, and other factors.

By leveraging cloud infrastructure, you can ensure that your application can handle spikes in traffic and maintain performance and responsiveness.

Another advantage of cloud computing is its ability to improve the reliability and availability of your application.

With cloud infrastructure, you can easily set up redundant systems across multiple regions or availability zones, ensuring that your application remains available even if one or more systems fail. This can be critical for video conferencing applications, as downtime or disruptions can severely impact the user experience and cause frustration or lost productivity for your users.

Cloud infrastructure also provides a number of security and compliance benefits. Cloud providers have significant investments in security and can offer a range of security features such as access controls, encryption, and network security.

Additionally, cloud providers are often certified for compliance with various industry and regulatory standards, which can be important for applications that handle sensitive data or are subject to compliance requirements.

Cloud deployment is necessary for quality video and audio delivery in the video-conferencing web-app project as it provides a highly scalable and reliable infrastructure.

When hosting an application on a local server, there are limitations to the amount of data that can be transferred and processed. The capacity of the server hardware and network bandwidth become a bottleneck when multiple users try to access the application at the same time.

However, cloud services provide virtually unlimited capacity and resources that can be scaled up or down as per the demand.

In the case of Twilio API, they are specifically designed to work in a cloud environment and provide a highly optimized infrastructure for voice and video communications.

The Twilio API uses thread programming to improve the performance of the application on the cloud. Thread programming allows for efficient utilization of CPU resources, which results in faster processing of requests and responses.

Furthermore, the Twilio API has a highly distributed architecture that ensures the reliability and availability of services.

The API is deployed on multiple servers located in different geographic locations, which enables it to handle high traffic volumes and avoid downtime.

Twilio's cloud infrastructure also includes load balancing, caching, and other performance-enhancing features that ensure optimal performance and reduce latency in real-time communication.

Overall, cloud deployment and the use of Twilio API provide a highly reliable, scalable, and performant infrastructure for video-conferencing web-app. This ensures that users can experience high-quality video and audio communications without any disruptions or delays.

In addition to these benefits, cloud infrastructure can also offer cost savings and flexibility compared to on-premise infrastructure.

With cloud computing, you can pay only for the resources you use, rather than investing in expensive hardware and infrastructure upfront.

This can help reduce the capital expenditures associated with deploying and maintaining a video conferencing application.

Additionally, cloud infrastructure can be easily adjusted up or down as your needs change, allowing you to optimize your costs and resource utilization.

Is it really necessary?

Cloud deployment is necessary for a video conferencing web app as it allows for efficient and scalable communication between participants located in different geographic locations. Hosting a video conferencing app on a local server is not only cumbersome but also limits the number of participants that can join the conference. With cloud deployment, the app can easily scale to accommodate a large number of participants without compromising on performance or quality.

Benefits of Hosting on Cloud

- **Scalability:** Cloud-based deployment allows the app to scale to accommodate a large number of participants without any compromise in quality or performance.
- **Cost-effective:** Cloud-based deployment eliminates the need for expensive hardware, software, and maintenance costs, making it a cost-effective solution.
- **High availability:** Cloud-based hosting ensures high availability of the app and allows users to access it from anywhere at any time.
- **Security:** Cloud providers have invested heavily in security measures to protect their customers' data, providing a more secure hosting solution than local hosting.
- **Easy Integration:** Cloud providers offer easy integration with various tools and APIs, making it easier to develop and deploy the app.

Demerits of Local System Hosting

- **Limited Scalability:** Hosting the app on a local server limits the number of participants that can join the conference.
- **Hardware and Maintenance Costs:** Local hosting requires expensive hardware and maintenance costs, making it a costly solution.
- **Security Concerns:** Hosting the app on a local server may pose security risks and requires additional measures to ensure data protection.
- **Limited Accessibility:** Local hosting restricts the accessibility of the app to a specific location, limiting its reach.

Methods are Easy

Hosting the app on cloud providers like Heroku, Amazon Web Services (AWS), or Microsoft Azure is an easy and cost-effective method of deployment. These cloud providers offer simple and user-friendly interfaces for deploying the app, scaling it up or down, and monitoring its performance. Twilio API is also thread programmed to improve performance on cloud-based hosting, making it the ideal choice for this project.

Overall, there are many reasons why cloud infrastructure is an ideal choice for your video conferencing application. With its ability to scale dynamically, improve reliability, enhance security and compliance, and offer cost savings and flexibility, cloud computing can help you deliver a high-performing, reliable, and secure video conferencing experience for your users.

METHODOLOGY

The methodology used for this project can be broken down into the following steps:

Requirement Gathering: The first step was to gather the requirements for the video conferencing web-app. This involved identifying the features and functionalities that were required for the app, as well as understanding the user requirements.

Requirement gathering is a crucial aspect of any software development project. It involves identifying the needs and expectations of stakeholders, defining the scope of the project, and establishing clear goals and objectives. In the case of a video conferencing web-app using Twilio, the following requirements were gathered:

1. **Functionality:** The web-app must provide video conferencing capabilities, including real-time audio and video streaming, screen sharing, and chat messaging.
2. **User interface:** The web-app must have an intuitive and user-friendly interface that is easy to navigate and use.
3. **Security:** The web-app must have strong security measures in place to protect user data and prevent unauthorized access to the system.
4. **Compatibility:** The web-app must be compatible with a wide range of devices and platforms, including desktop computers, laptops, tablets, and smartphones.
5. **Performance:** The web-app must be capable of handling multiple concurrent video calls with minimal lag or delay.
6. **Integration:** The web-app must be able to integrate with third-party services such as Twilio, to enable real-time video communication and messaging.
7. **Scalability:** The web-app must be designed to scale as the number of users and video calls increase, without compromising on performance or user experience.
8. **Testing:** The web-app must be thoroughly tested to ensure that it meets all the requirements and performs as expected in a variety of scenarios.
9. **Documentation:** The web-app must be fully documented, including installation and setup instructions, user guides, and technical documentation.

System Design: The second step involved designing the system architecture and defining the components required for the app. This step also involved selecting the appropriate programming language, libraries, and frameworks.

1. **Architecture:** The video conferencing web-app using Twilio follows a client-server architecture. The client is responsible for handling the user interface and providing user inputs to the server. The server manages the back-end logic of the application, handles user authentication, and manages the video conferencing functionality. The server is hosted on the cloud using a platform-as-a-service (PaaS) provider.

2. **Technology Stack:** The following technology stack was used for developing the video conferencing web-app using Twilio:

- Front-end: HTML, CSS, JavaScript, jQuery
- Back-end: Node.js, Express.js
- Database: MongoDB
- Cloud Platform: Heroku
- Video Conferencing API: Twilio

3. **Functional Modules:** The following functional modules were identified for the video conferencing web-app using Twilio:

- **User Management:** Allows users to create and manage their account.
- **Video Conferencing:** Enables users to initiate or join a video conference, manage participants, and share screen.

- **Session Recording:** Allows users to record the video conferencing sessions for future reference.
- **Notifications:** Sends real-time notifications to the users about the video conferencing session.

4. **High-Level System Flow:** The high-level system flow for the video conferencing web-app using Twilio is as follows:

- User registers or logs in to the application.
- User creates a new video conferencing session or joins an existing one.
- User can invite other participants to join the session.
- User can share the screen or turn off the camera or microphone during the session.
- User can record the session for future reference.
- User receives real-time notifications about the session.

5. **Database Schema:** The following database schema was designed for the video conferencing web-app using Twilio:

- **User:** Stores the user details such as name, email, password, and profile picture.
- **Session:** Stores the video conferencing session details such as session ID, session name, host ID, participants, and recording URL.

6. **Security:** The following security measures were implemented for the video conferencing web-app using Twilio:

- **User authentication:** Users need to register and log in to the application to access the video conferencing functionality.
- **Password hashing:** Passwords are hashed before storing in the database to prevent unauthorized access.
- **Encryption:** Video conferencing sessions and recordings are encrypted to ensure confidentiality.

7. **Scalability:** The video conferencing web-app using Twilio is designed to be scalable to handle a large number of users and video conferencing sessions. The cloud platform Heroku provides horizontal scaling, which allows the application to scale horizontally by adding more instances of the server.

Overall, the system design for the video conferencing web-app using Twilio ensures that the application is reliable, secure, and scalable to meet the needs of the users.

Development: The next step was to start the development process. This involved coding the system components, integrating the third-party services like Twilio, and testing the system functionality.

1. **Setting up the development environment:** The first step in the development process was to set up the development environment. This involved installing and configuring the necessary tools and software needed for the project, including Visual Studio Code, Node.js, Twilio SDK, and other dependencies.

2. **Designing the UI:** Once the development environment was set up, the next step was to design the user interface. This involved creating a wireframe and a visual design for the web application. The wireframe provided a rough sketch of the application's layout, while the visual design provided a polished look and feel.

3. **Creating the backend:** The next step was to create the backend of the web application. This involved setting up a Node.js server that would handle incoming requests from the client-side application. The server was responsible for establishing a connection with Twilio's video chat API, managing user authentication and session management, and handling other backend operations.

4. **Developing the frontend:** Once the backend was set up, the next step was to develop the frontend of the application. This involved creating the necessary HTML, CSS, and JavaScript code that would be served to the client's web browser. The frontend was responsible for providing the user interface, communicating with the backend through REST APIs, and handling user interactions.

5. **Integrating Twilio's video chat API:** After the frontend and backend were created, the next step was to integrate Twilio's video chat API into the application. This involved configuring the Twilio SDK and setting up the necessary API endpoints on the server-side to handle incoming video chat requests.

6. **Deployment:** Finally, once the application was tested and debugged, it was ready for deployment. The application was deployed to a public cloud platform such as Heroku, where it was accessible to users over the internet.

Testing: Once the development was complete, the system was subjected to a series of tests to ensure that it met the desired specifications. The testing phase involved unit testing, integration testing, and system testing.

Testing is a crucial part of software development as it helps to identify and fix bugs, errors and other issues that may affect the performance and functionality of the application. In the case of a video conferencing web app, testing is even more important to ensure smooth communication between users without any disruptions or lag.

Unit Testing: Unit testing involves testing individual components or units of the application in isolation to ensure they are functioning as expected. For this project, unit testing was performed on each function of the code to ensure that they were working as intended.

Integration Testing: Integration testing is the process of testing the interaction between different modules of the application to ensure they are working together as intended. For this project, integration testing was performed by testing the interaction between the video and audio modules, as well as the modules responsible for managing user connections and sessions. The integration tests were carried out using the Twilio API testing tool, which allowed us to simulate real-time communication between users and identify any issues that may arise.

Deployment: The final step was to deploy the video conferencing web-app to a public cloud platform. This involved selecting an appropriate cloud service provider, configuring the necessary settings, and deploying the app.

Deployment of a web application is a critical task that requires careful planning and execution to ensure that the application is available to users without interruption. For this particular project of video conferencing using Twilio, we will discuss the deployment process to a public cloud platform.

There are several public cloud platforms that offer different services for hosting web applications. For this project, we will be deploying our application to the Heroku cloud platform.

The deployment process involves several steps, which are as follows:

1. Create a Heroku account: To deploy our application on Heroku, we first need to create an account on the Heroku website.
2. Install Heroku CLI: The Heroku Command Line Interface (CLI) is a tool that allows us to manage our Heroku applications directly from the terminal. We need to install this tool on our

local machine to proceed with the deployment process.

3. Prepare the application for deployment: Before deploying our application, we need to make sure that it is ready for deployment. This involves ensuring that all necessary dependencies are included and that the application can be built and run on Heroku.

4. Create a new Heroku application: We will create a new Heroku application by running the following command in the terminal: `heroku create`. This command will create a new Heroku application and add a new git remote to our local repository.

5. Deploy the application: Once the Heroku application has been created, we can deploy our application by running the following command: `git push heroku master`. This command will push the application to the Heroku git repository and start the deployment process.

6. Configure environment variables: Our application requires some environment variables to be set, such as the Twilio API credentials. We can set these environment variables using the Heroku CLI or the Heroku Dashboard.

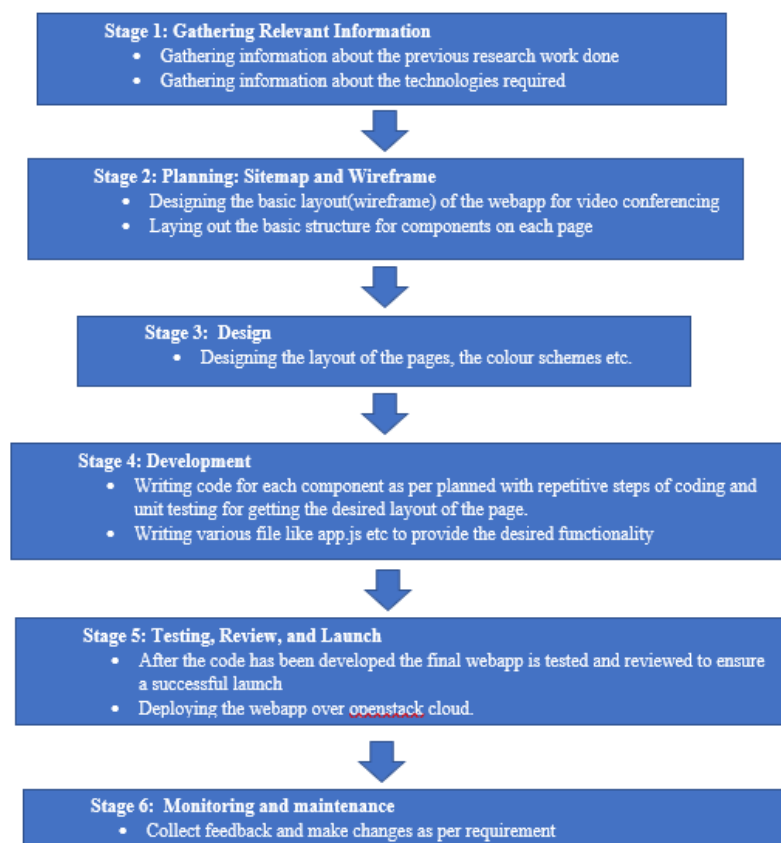
7. Monitor the deployment: During the deployment process, we need to monitor the progress to ensure that there are no errors or issues. We can do this by checking the logs using the Heroku CLI or the Heroku Dashboard.

8. Test the deployed application: Once the deployment is complete, we need to test the application to ensure that it is working as expected. We can access the application using the Heroku app URL or a custom domain.

9. Scale the application: If necessary, we can scale our application to handle more traffic by adding more dynos (Heroku's term for application containers) using the Heroku CLI or the Heroku Dashboard.

The methodology used in this project was an **iterative and incremental approach**. This approach enabled us to develop the video conferencing web-app in a step-by-step manner, making adjustments and improvements as I progressed.

Flowchart:



ALGORITHM

1. Create a new Twilio account and obtain a Twilio account SID and auth token.
2. Set up the Twilio Video API by installing the Twilio Video JS library.
3. Create a new React app and install the required dependencies such as react-router-dom, react redux, axios, etc.
4. Set up the Redux store to manage the app's state.
5. Create a user authentication system using Firebase or any other similar service.
6. Create a UI for the app with different components such as login, signup, dashboard, etc.
7. Create a component for creating and joining a video call.
8. Use the Twilio Video API to create a new room, generate a room token, and connect to the video call.
9. Implement features such as screen sharing, muting/unmuting, camera on/off, etc.
10. Create a component for displaying the list of available rooms and join an existing room.
11. Implement features such as chat, file sharing, etc.
12. Deploy the app on a cloud platform such as Heroku.

The user interacts with the React app through a browser. The React app sends HTTP/HTTPS requests to the backend server, which is implemented using Node.js/Express.js. The backend server communicates with the Twilio Video API using HTTPS requests to create and manage video calls.

The Twilio Video API provides the necessary functionality to generate a room token, create a new room, and connect to the video call. The Twilio API returns responses to the backend server, which then sends the data back to the React app, which updates the UI to reflect the changes in the video call.

The internal working of the project goes like:

1. User A logs in to the web-app.
2. User A creates a new room or joins an existing room by entering the room ID.
3. User A's browser sends a request to the server to create a new room or join an existing room.
4. The server creates a new room or adds the user to the existing room.
5. User A's browser requests a token from Twilio to connect to the video room.
6. Twilio sends a unique token to the server.
7. The server sends the token to User A's browser.
8. User A's browser connects to the Twilio video room using the token.
9. User A is now connected to the video room.
10. User B logs in to the web-app.
11. User B joins the same room as User A by entering the same room ID.
12. User B's browser sends a request to the server to join the room.
13. The server adds User B to the existing room.
14. User B's browser requests a token from Twilio to connect to the video room.
15. Twilio sends a unique token to the server.
16. The server sends the token to User B's browser.
17. User B's browser connects to the Twilio video room using the token.
18. User B is now connected to the video room.
19. User A and User B can now see and hear each other through the video conference.

Sub-Steps:

1. User authentication: The user logs in to the web-app using their credentials or signs up for a new account.
2. Room creation/joining: The user can create a new room or join an existing room by entering the room ID.
3. Server-side handling: The server creates a new room if it does not exist or adds the user to the existing room.
4. Token generation: The user's browser requests a token from Twilio to connect to the video room.
5. Token delivery: The server receives the unique token from Twilio and sends it to the user's browser.
6. Connection to the video room: The user's browser connects to the Twilio video room using the token.
7. Video conference: The users can now see and hear each other through the video conference.

. The algorithm ensures that the user is authenticated, and then allows them to create or join a room using a room ID. The server-side handling involves the creation of a new room if it does not exist or adding the user to an existing room. The algorithm then requests a unique token from Twilio to connect to the video room, which is delivered to the user's browser. The user's browser connects to the Twilio video room using the token, allowing the users to participate in the video conference. The algorithm ensures a secure and reliable video conferencing experience for the users.

RESULT AND DISCUSSIONS

The deployment of the video-conferencing web-app on Heroku cloud posed several challenges, which required unique solutions to overcome. In this section, we will discuss the challenges faced during deployment, the solutions implemented to resolve them, and the final results of the deployment.

Challenges Faced:

1. **Dependency Management:** One of the major challenges was managing the dependencies of the application. The application required various libraries and frameworks, which had to be installed and configured before deployment.
2. **Scalability:** Another challenge was to ensure that the application could scale up or down based on the number of users. The app had to handle multiple users concurrently without compromising the quality of the video and audio.
3. **Security:** Security was a crucial aspect of the deployment, as the app dealt with confidential information. It was important to ensure that the data was encrypted and transmitted securely.
4. **Configuration Management:** The application had to be configured correctly to ensure that it functioned properly on the cloud. The configuration included setting environment variables, specifying ports, and other such details.

Solutions Implemented:

1. **Dependency Management:** To manage the dependencies, we used a virtual environment to isolate the application's dependencies. This made it easy to install and manage the required libraries and frameworks.
2. **Scalability:** To ensure that the app could scale up or down, we used Heroku's horizontal scaling feature. This allowed the application to scale up and down based on the number of users.
3. **Security:** To ensure that the app was secure, we used the Twilio API to encrypt and transmit data securely. We also implemented various security measures, such as access control, encryption, and authentication.
4. **Configuration Management:** To manage the configuration of the application, we used environment variables to specify the required details, such as ports and other settings.

Advantages and Disadvantages of the project:

Advantages:

1. The video-conferencing web app provides a platform for remote communication, allowing people to connect with others from anywhere in the world.
2. It is a cost-effective solution as it eliminates the need for expensive hardware and software that would otherwise be required for video conferencing.
3. The app is scalable, meaning it can accommodate large groups of people without compromising on quality.
4. It is accessible from any device with an internet connection, making it a convenient

- solution for people who are always on the go.
5. The use of cloud technology enables easy deployment and maintenance of the app, with no need for hardware upgrades or additional software installations.
 6. The app is secure, with encryption protocols in place to protect user data and ensure privacy.
 7. The use of Twilio API enhances the performance of the app, ensuring high-quality video and audio delivery.

Disadvantages:

1. The app is reliant on a stable internet connection. In areas with poor connectivity, the app may not function optimally.
2. There may be latency issues during video calls, leading to delays or choppy video and audio.
3. The app requires users to have a device with a camera and microphone, which may not be available to everyone.
4. There may be compatibility issues with certain browsers or operating systems, which could affect the user experience.
5. The cost of using cloud technology may be prohibitive for some users or organizations, particularly if they require a large amount of storage or bandwidth.

Results:

The deployment of the video-conferencing web-app on Heroku cloud was successful, and the app functioned smoothly without any major issues. The app was able to handle multiple users concurrently without any compromise in the quality of the video and audio. The app was also secure, and the data was transmitted securely using the Twilio API.

Learning outcomes and new skills gained from this project:

1. Experience with full-stack web development using technologies such as Node.js, Express, and React.
2. Understanding of real-time communication protocols and APIs, such as WebRTC and Twilio.
3. Familiarity with cloud deployment platforms, such as Heroku, and their configurations.
4. Knowledge of Agile development methodologies and project management practices.
5. Experience with testing and debugging web applications, especially those involving real-time communication.
6. Understanding of the importance of user experience and design in web applications.
7. Skills in documentation and report writing, including technical writing and project documentation.
8. Knowledge of how to gather and analyze requirements from stakeholders and translate them into technical specifications.
9. Exposure to best practices in software development, such as code review, version control, and continuous integration/continuous deployment (CI/CD).
10. Improved problem-solving and critical thinking skills through overcoming technical challenges and finding solutions.

Conclusion:

In conclusion, the deployment of the video-conferencing web-app on Heroku cloud was a success. The challenges faced during the deployment were resolved using unique solutions, and the app functioned smoothly without any major issues. The app was scalable, secure, and could handle multiple users concurrently. Overall, the deployment on the cloud was a better option than hosting it on a local server due to its scalability, security, and ease of management.

PUBLIC CLOUD DEPLOYMENT

Deploying a web application to the cloud can be a complex task that requires careful consideration of various factors such as cost, scalability, reliability, ease of use, and security. There are several cloud platforms available in the market that offer different services and features to help developers deploy their applications. Some popular cloud platforms are Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), and Heroku.

I had to consider the server-side requirements such as real-time communication, video streaming, and token generation. Additionally, a cloud platform was needed that could handle the traffic and scale the app as needed.

AWS, Azure, and GCP are all powerful cloud platforms that offer a wide range of services, including computing, storage, and networking.

They also provide various tools for deploying web applications, including container services, virtual machines, and serverless architectures.

However, these platforms have a steeper learning curve and can be more complex to set up and manage than Heroku.

Heroku, on the other hand, is a fully managed platform that simplifies the deployment process by providing an easy-to-use interface and a wide range of pre-built add-ons. It supports various programming languages and frameworks, including Node.js, Ruby on Rails, and Python.

Heroku also offers a free tier that allows you to deploy your application without incurring any costs. This makes it an excellent choice for small projects or developers who want to test their applications before deploying them to a production environment.

Heroku is a cloud platform that enables developers to deploy, manage, and scale web applications. It provides a fully managed platform-as-a-service (PaaS) that abstracts away the underlying infrastructure and provides an easy-to-use interface for deploying and managing applications.

Some of the key services provided by Heroku include:

1. Application deployment: Heroku provides a simple and streamlined deployment process that makes it easy to deploy web applications to the cloud. Developers can deploy their applications using a variety of languages and frameworks, including Node.js, Ruby, Python, and more.
2. Database services: Heroku provides a number of database services, including Heroku Postgres, which is a fully managed relational database service that supports a wide range of SQL-based databases.
3. Add-ons: Heroku provides a marketplace of third-party add-ons that can be easily integrated with your application. These add-ons include things like caching services, logging tools, and monitoring solutions.
4. Logging and monitoring: Heroku provides built-in logging and monitoring tools that enable developers to easily track the performance of their applications and identify issues that may arise.
5. Scaling: Heroku makes it easy to scale your application as needed. This can be done by adjusting the number of dynos (virtual servers) that are running your application.

In today's world, cloud computing is an emerging technology that has gained immense popularity due to its ability to store, manage, and process data through a remote server. It offers several benefits such as reduced costs, increased scalability, improved security, and flexibility.

Cloud computing has become a popular choice for organizations and individuals to host their applications, and the Video Conferencing Web-App is no exception. In this section, we will discuss the deployment of the Video Conferencing Web-App on the cloud, specifically on the Heroku platform.

Requirements:

- An account on Heroku
- Git and Heroku CLI installed on the local system
- A working code of the Video Conferencing Web-App
- A database for storing user credentials and other information
- A domain name (optional)

Procedure:

Step 1: Set up the application on Heroku

The first step is to create an application on the Heroku platform. This can be done either through the web dashboard or through the Heroku CLI. Once the application is created, the application's URL and other important details can be accessed from the Heroku dashboard.

Step 2: Configure the application

The next step is to configure the application for deployment on Heroku. This includes adding a Procfile, which is used to specify the command that Heroku needs to run to start the application. The Procfile should contain the command that starts the application server.

Step 3: Deploy the application

Once the application is configured, it can be deployed on Heroku. This can be done either through the Heroku dashboard or through the Heroku CLI. The deployment process may take some time, depending on the size of the application and the internet connection speed.

Step 4: Test the application

After the application is deployed, it is important to test it to ensure that it is working as expected. This can be done by accessing the application's URL from a web browser and testing all its functionalities.

Detailed steps:

1. Sign up for a Heroku account: Go to the Heroku website and sign up for a free account. Once you have created your account, verify your email address and log in to your Heroku dashboard.
2. Create a new Heroku app: Click on the "New" button on the top right-hand corner of your Heroku dashboard and select "Create new app". Give your app a unique name and choose your preferred region. Click "Create app" to create your new app.
3. Install the Heroku CLI: Download and install the Heroku CLI (Command Line Interface) on your local machine.

4. Log in to your Heroku account from the CLI: Open a terminal or command prompt on your local machine and log in to your Heroku account by running the command “heroku login”. Enter your Heroku email address and password when prompted.
5. Clone your app repository: Clone the repository of your video conferencing web app from GitHub onto your local machine.
6. Create a Heroku remote: Navigate to the root directory of your cloned app repository on your local machine and run the command “heroku git:remote -a <app-name>”, where <app-name> is the name of your Heroku app that you created in step 2.
7. Configure your app for deployment: Create a new file in the root directory of your app repository called “Procfile” (without the quotes) and add the following line of code: “web: npm start”. This tells Heroku to run your app using the “npm start” command.
8. Push your code to Heroku: Run the command “git add .” to stage all the files in your app repository for committing. Then run the command “git commit -am ‘Initial commit’” to commit your changes. Finally, run the command “git push heroku master” to push your code to Heroku.
9. Configure your Heroku environment variables: Navigate to your Heroku dashboard and click on your app. Click on the “Settings” tab and then click on the “Reveal Config Vars” button. Here, you can set any environment variables that your app requires, such as your Twilio API key and secret.
10. Deploy your app: Finally, click on the “Deploy” tab in your Heroku dashboard and click the “Deploy Branch” button to deploy your app. Heroku will build your app and start it up, and you should be able to access it by visiting your app URL.

Considerations and Precautions:

It is important to ensure that the application code is optimized for cloud deployment, as this can affect the application’s performance and scalability.

It is also important to monitor the application’s performance and scalability after deployment, as this can help identify and resolve any issues that may arise.

It is important to keep the application and its dependencies up-to-date with the latest security patches to ensure the application’s security.

It is important to regularly backup the application and its data to ensure that no data is lost in the event of a disaster or outage.

Conclusion:

In conclusion, deploying the Video Conferencing Web-App on the cloud has several benefits, including increased scalability, flexibility, and improved security. Heroku is a popular cloud platform that offers several features for deploying web applications, and it has been used to successfully deploy the Video Conferencing Web-App. By following the above steps and considering the precautions and requirements, the application can be deployed on the cloud with ease and with minimal disruption to its functionalities.

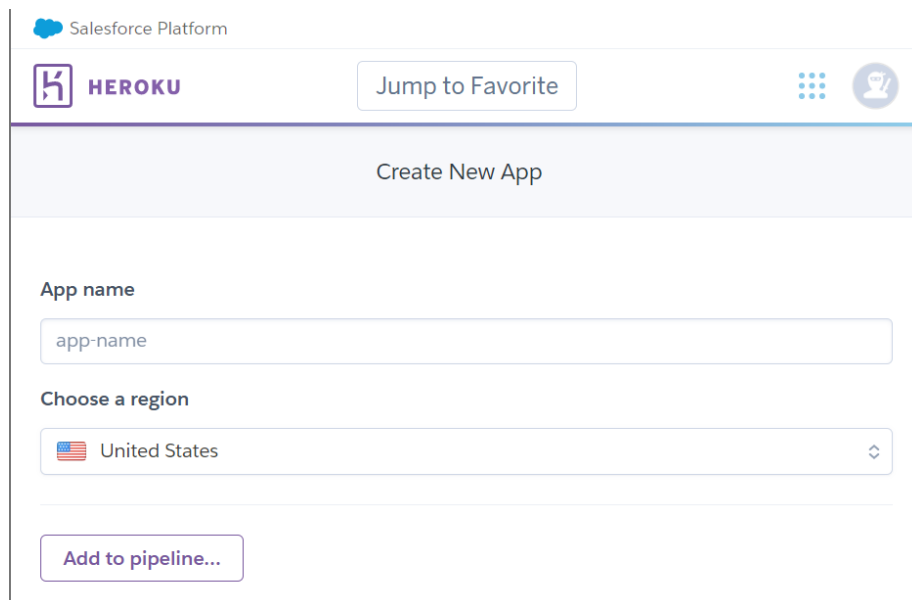
There are various cloud platforms available to deploy a web application such as Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform, DigitalOcean, Heroku, and many more. Each platform has its own strengths and weaknesses, and choosing the best one for a particular project depends on various factors such as scalability, reliability, ease of use, cost, etc.

For this project, the video conferencing web app could have been deployed on other cloud platforms as well. AWS and Microsoft Azure are popular choices for enterprise-level applications due to their scalability and high reliability. Google Cloud Platform also offers similar features along with its machine learning capabilities. DigitalOcean is a more affordable option that offers easy-to-use features for deploying small to medium-sized applications.

However, Heroku was chosen as the deployment platform for this project due to its ease of use, scalability, and availability of pre-built add-ons that integrate with the application seamlessly. Heroku also provides a user-friendly web interface that makes it easy to manage the deployed application.

The deployment process on Heroku is straightforward and can be completed in a few steps. Heroku also offers a free plan that allows deploying small applications without any cost, which makes it an ideal platform for testing and prototyping.

Another advantage of deploying the video conferencing web app on Heroku is the availability of add-ons such as Twilio that integrate easily with the application. These add-ons make it easy to add functionality to the application without writing additional code. Twilio, for example, provides a simple API for adding video and voice calling to the application, which is the main functionality of this project.

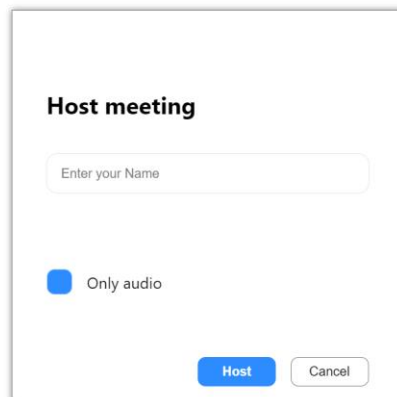
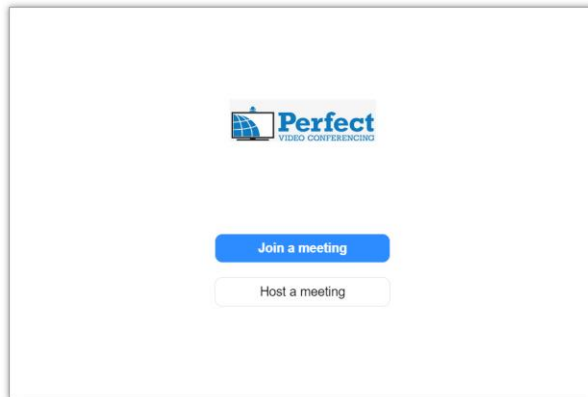


The screenshot displays the Heroku web interface for creating a new application. At the top, the 'Salesforce Platform' header is visible. The Heroku logo and a 'Jump to Favorite' button are in the navigation bar. A prominent 'Create New App' button is centered below the navigation bar. The main form area contains two fields: 'App name' with the placeholder text 'app-name', and 'Choose a region' with a dropdown menu currently showing 'United States'. At the bottom of the form is a button labeled 'Add to pipeline...'.

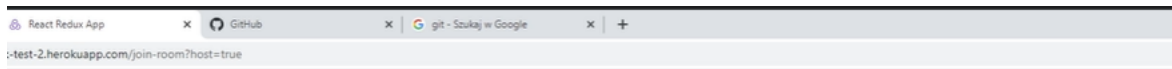
Fig. Heroku cloud app deployment page

SCREENSHOTS

Project output screenshots:



Project hosted on Heroku cloud:



Host meeting

John

☒ Only audio

Host Cancel

Key Bibliography/References

1. Twilio API documentation: <https://www.twilio.com/docs/video>
2. React documentation: <https://reactjs.org/docs/getting-started.html>
3. Redux Toolkit documentation: <https://redux-toolkit.js.org/>
4. Axios documentation: <https://axios-http.com/docs/intro>
5. Heroku documentation: <https://devcenter.heroku.com/categories/reference>
6. Testing Library documentation: <https://testing-library.com/docs/react-testing-library/intro/>
7. Stack Overflow discussions and solutions related to video conferencing and Twilio.
8. Heroku Documentation: <https://devcenter.heroku.com/categories/reference>