A Project On

Multimedia Over IP

Submitted in fulfillment of the course named

CS656 - Internet and Higher Layer Protocols

at

Master of Science

In

Department of Computer Science

By

Chaitanya Karnati(ck338)

Satya Surya Kesava Srikar Palivela(sp2878)

Lokesh Mekala(lm455)

Contents

1 Introduction

The ability to hold meetings via video chat is an essential component of the modern world. However, due to the intricacy of the system, the vast majority of developers struggle when attempting to apply it. People now have the option to work from the comfort of their own homes as a result of the pandemic. On the other side, this calls for highly effective video conferencing and handling of packets. Both React.js and WebRTC are fantastic frame- works for the building of applications that facilitate web-based video conferencing. By creating a handler for video conferencing, we will conduct a comprehensive investigation into these frameworks.

2 Prerequisites

To follow along, the reader should have some basic knowledge of the following:

Getting started with React.js ES6
Getting started with Node.js and Command Terminal
Getting started with Graphql and WebRTC

3 Fundamentals of web-based video conferencing

Video conferencing is the visual interaction between two or more nodes connected to the internet. It supports the transmission of static images, texts, full-motion, and high-definition audio between multiple nodes.

3.1 WebRTC

WebRTC is an open-source technology that provides real-time communication capabilities to an application. It supports video, audio and other kinds of data to be transferred between nodes.

In other words, it enables developers to integrate voice and video functionalities into their applications.

3.2 Components of video conferencing system

Web-based video conferencing involves the synergy of various frameworks and libraries which include the following:

Voice over Internet Protocol (VoIP) and Integrated Service Digital Networks (ISDN).
Microphones and webcams.
Display screen or projector.
Software-based coding and decoding technologies (CODEC).
Acoustic Echo Cancellation (AEC) software for audio optimization and real-time communication.

3.3 Requirements for web packets handling

For network communication to succeed, it is necessary to have a unified standard for defining the architecture of communication systems. A digital environment supporting multiple data types including audio and video significantly increases the efficiency of a video conferencing application including greater bandwidth utilization.

3.4 Implementation and coding

The application is a full-stack project that is divided into two segments:
The client-side
The server-side

4 Client-side setup - (index.js)

The client interface is set up using React.js which is a lightweight frontend Javascript library. The various pages of the client interface include the following:

Step 1: Getting started with a new React app npx create-react-app react-video-conferencing-app The command above should get you started with a new React app with all the default dependencies installed.

cd react-video-conferencing-app npm start The command above will change the directory to your new react app and start the development server.

Step 2: Installing the required client dependencies For successful development, a few dependencies must be installed. They enable the React app, as well as perform specific instructions.

We install the dependencies by running npm install or yarn add to initialize an empty Node.js project in the terminal.

Add the following dependencies in the package.json file and then run npm install to download them.

```
"apollo-cache-inmemory": "^1.1.9",
"apollo-client": "^2.2.5",
"apollo-client-preset": "^1.0.8",
"apollo-link-http": "^1.4.0",
"apollo-link-schema": "^1.0.6",
"apollo-link-ws": "^1.0.7",
"apollo-utilities": "^1.0.10",
"classnames": "^2.2.5",
"react-apollo": "^2.0.4",
"react-dom": "^16.2.0",
"react-redux": "^5.0.7",
"react-router": "^4.2.0",
"react-router-config": "^1.0.0-beta.4",
"react-router-dom": "^4.2.2",
"react-stay-scrolled": "^2.1.1",
"redux": "^3.7.2",
"redux-actions": "^2.2.1",
"redux-devtools-extension": "^2.13.2",
"redux-thunk": "^2.2.0",
"socket.io": "^2.4.0",
"socket.io-client": "^2.0.4",
"socket.io-redis": "^5.2.0",
"socketio-jwt": "^4.5.0",
"style-loader": "^0.20.2",
```

Step 3: Setting up the client index file This is the main file for integrating the client and the server code. It enables the initialization of the React DOM element, Apollo- Client elements, and the WebRTC adapter.

```
import 'webrtc-adapter';
import { InMemoryCache } from 'apollo-cache-inmemory';
import { ApolloClient } from 'apollo-client';
import { split } from 'apollo-client-preset';
import { HttpLink } from 'apollo-link-http';
import { WebSocketLink } from 'apollo-link-ws';
import { getMainDefinition } from 'apollo-utilities';
import React from 'react';
import { ApolloProvider } from 'react-apollo';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import { renderRoutes } from 'react-router-config';
import { BrowserRouter } from 'react-router-dom';
import './styles/index.scss';
import routes from './routes';
import store from './store';
import { setToken } from './actions/token';
store.dispatch(setToken(window.__JWT_TOKEN__));

const httpLink = new HttpLink({
  uri: process.env.GRAPHQL_URI,
  credentials: 'same-origin',
});

const wsLink = new WebSocketLink({
  uri: process.env.GRAPHQL_WS_URI,
  options: {
    reconnect: true,
  },
```

Step 4: Setting up client-side routes and pages The application has five major pages: Home Page Login/ Sign-Up Page Contacts Page Message Page Settings Page And their respective routes are implemented as follows:

```
import React from 'react';
import { Redirect } from 'react-router';
import {
  INDEX_ROUTE,
  LOGIN_ROUTE,
  SIGNUP_ROUTE,
  CONTACTS_ROUTE,
  MESSAGES_ROUTE,
  CONTACT_REQUESTS_ROUTE,
  SETTINGS_ROUTE,
} from '../constants';

import PageLayout from '../containers/PageLayout';
import Login from '../containers/Login';
import Signup from '../containers/Signup';
import Contacts from '../containers/Contacts';
import Messages from '../containers/Messages';
import Settings from '../containers/Settings';

export default [{
  component: PageLayout,
  routes: [
    { path: INDEX_ROUTE, exact: true, component: () => <Redirect to={
    { path: LOGIN_ROUTE, component: Login },
    { path: SIGNUP_ROUTE, component: Signup },
    { path: CONTACTS_ROUTE, component: Contacts },
    { path: MESSAGES_ROUTE, component: Messages },
    { path: SETTINGS_ROUTE, component: Settings },
  ],
}];
```

Step 5: Setting up the video components The video component is essential since it facilitates the connection and communication between various nodes in the application. It also attaches event listeners to the microphone and webcam of the connected devices.

The video component enables the following operations:
Call status
Accept call
Ignore call
Hang up
The implementation of the video component is illustrated below:

```
import React from 'react';
import PropTypes from 'prop-types';
import { connect } from 'react-redux';
import classNames from 'classnames';

import { preferOpus } from '../helpers/sdp-helpers';
import {
  CallStatuses,
  acceptCall,
  ignoreCall,
  handleIceCandidate,
  sendSessionDescription,
  setCallStatusToInCall,
  setCallStatusToAvailable,
  setCallStatusToHangingUp,
  emitHangup,
} from '../actions/call';

import { addError } from '../actions/error';
import Available from '../components/VideoChat/Available';
import Calling from '../components/VideoChat/Calling';
import ReceivingCall from '../components/VideoChat/ReceivingCall';
import Controller from '../components/VideoChat/Controller';
import CallOverlay from '../components/VideoChat/CallOverlay';
import BannerContainer from '../components/Layout/BannerContainer';

 startPeerConnection() {
    try {
      this.peerConnection = new RTCPeerConnection({
        iceServers: this.props.iceServerConfig,
```

5 The server-side setup

5.1 Installing the required server dependencies

For Node.js to perform the required server operations, some dependencies must be installed in the server folder.

We do this by running the command below: npm install

5.2 Setting up the server – (index.js)

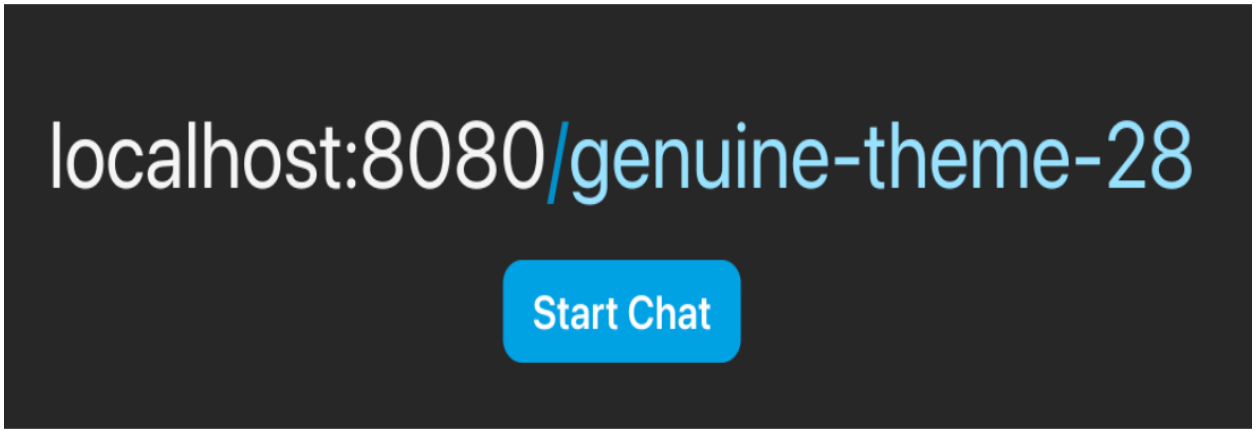
The backend server is set up using Node.js and Graphql. To guarantee optimal performance, it is essential to have a robust server instance.

5.3 Setting-up Graphql server

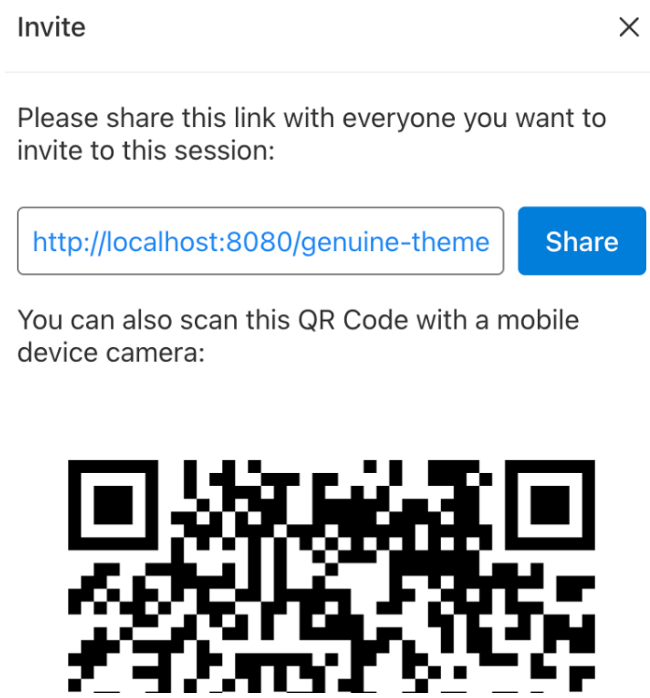
The Graphql server acts as an interface between the client and the server to provide a robust mechanism for video and audio data transfer.

Graphql servers are fully equipped with mutations for data modification and alteration, query for data fetching, and subscription for real-time data instance monitoring.

6 User Interface of App



- The above screenshot is the main page that creates a specified link to access the video conference.



- The above screenshot is the link sharing interface that we can share to the people to join the conference and it may generate a random QR Code for every call that is easy for the people to join.

7 Conclusion

This project explained the fundamentals, components, and requirements for implementing a web-based video-conferencing application. We also discussed error handling and testing.

8 References

- https://www.telestream.net/video/solutions/how-video-over-ip-works.htm
- https://github.com/topics/internet-radio
- https://github.com/GBR-Suppe/briefing-Master