

# CHAT APPLICATION USING AWS

## A MINI-PROJECT REPORT

*Submitted by*

*SAKSHI SHRUTI*

*RA2011003010924*

*ANURAG RISWADKAR*

*RA2011003010930*

*PRIYAL MITTAL*

*RA2011003010933*

*ARPITA SINGH*

*RA2011003010937*

*Under the guidance of*

**Mrs. Viji D**

(Associate Professor, Department of Computer Science & Engineering)

*in partial fulfillment for the award of*

*the degree of*

**BACHELOR OF TECHNOLOGY**

in

**COMPUTER SCIENCE & ENGINEERING**

of

**FACULTY OF ENGINEERING AND TECHNOLOGY**



**S.R.M. Nagar, Kattankulathur, Kancheepuram District**

# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

## BONAFIDE CERTIFICATE

Certified that this project report titled “Chat Application Using AWS” is the bona fide work of [“Arpita Singh”, “Sakshi Shruti”, “Priyal Mittal”, “Anurag Riswadkar”], who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Mrs. VIJI D  
Professor  
Department of Computer & Science Engineering

## TABLE OF CONTENTS

1.	ABSTRACT	4
2.	INTRODUCTION	5
3.	LITERATURE SURVEY	6
4.	REQUIREMENT ANALYSIS	10
5.	ARCHITECTURE	11
6.	IMPLEMENTATION	13
7.	RESULT	15
8.	CONCLUSION	16
9.	REFERENCES	20

# 1. ABSTRACT

Cloud Computing in recent times has unfolded as the most popular and efficient standard for managing and delivering useful services over the internet. Out of various cloud services and techniques, serverless computing represents an evolution of cloud-based programming technologies, and is an attestation to the growth and large-scale adoption of cloud concepts. However, aside from gaining agility and scalability, infrastructure costs are a major hindrance for companies following this pattern due to the increasing server loads and the need to increase server space. This is where serverless computing and services like AWS Lambda comes into picture. The project delves into the functioning of AWS Lambda along with other existing AWS services through the development of a serverless chat application which supports scalability without the addition of new servers.

## 2. INTRODUCTION

Organizations such as WhatsApp, Instagram and Facebook have long been promoting conversational UI (User Interface) based applications that are based on either text or voice or both. This has recently resulted in growing interest regarding how relevant technologies could be used to improve business in wide industry domains. The latest trend in cloud computing constructs is the use of serverless architecture. This fast-developing cloud model includes the provision of a server and its managerial operations by the cloud provider itself which eliminates the need for the user to maintain, monitor and schedule servers. This greatly simplifies the methodology of deploying the code into production process. Serverless code is employed in conjunction with code deployed in generic code designs, like microservices. Majority of the serverless applications run in state-less compute blocks that are triggered by events. The pricing depends on the number of executions rather than an already purchased number of compute capacity servers.

The purpose of this project is to implement a chat application that was built based on Amazon Web Services' Lambda framework which as mentioned above, is a rapidly evolving serverless computing platform, to enable users to send fast, real-time messages using a cloud service platform instead of a server.

### 3. LITERATURE SURVEY

#### 1) **Android chatting application based on wifi technology**

**Year:** 2016

**Publisher:** IJESRT

Technology trends in both hardware and software have driven the hardware industry towards smaller, faster and more capable mobile hand-held devices that can support a wider-range of functionality and open source operating systems. Mobile hand-held devices are popularly called smart gadgets. Adding text messaging functionality to mobile devices began to gain traction in the mobile communication services community in the early 1980s. The first action plan of the Group GSM was approved in December 1982, requesting "The services and facilities offered in the public switched telephone networks and public data networks should be available in the mobile system". This plan included the exchange of text messages either directly between mobile stations, or transmitted via Message Handling Systems widely in use at that time. The first proposal which initiated the development of exchanging information or sent message to the user was made by a contribution of Germany and France into the GSM group meeting in February 1985 in Oslo. Initial growth was slow, with customers in 1995 sending on average only 0.4 messages per GSM customer per month. In 2013, 6.1 trillion text messages were sent. This translates into 193000 SMS per second. While SMS reached its popularity as a person-to-person messaging, another type of SMS is growing fast: application-to-person (A2P) messaging. A2P is a type of SMS sent from a subscriber to an application or sent from an application to a subscriber. It is commonly used by financial institutions, airlines, hotel booking sites, social networks, and other organizations sending SMS from their systems to their customers. According to research in 2013, A2P traffic is growing faster than P2P messaging traffic. Over the years several approaches and solutions presented considering the secure exchanging of message thorough client and webserver. The various researches have been done and are going on location based project and in the same ratio various applications have been developed on location-based and message sharing system. As the amount of user deal to exchange the information with other people to store the large amount of data the

existing system cannot support the centralized database. Sensitive data may also be leaked accidentally due to improper disposal or resale of storage media. Diesburg et al. surveys, summarizes and compares existing methods of providing confidential storage of data but it cannot support the secured communication between the user application and the webserver. Ramesh Shrestha, Yao Aihong et al. developed location and message sharing system for Android Platform. In the present location based services, the works are mainly focused on how to handle the location, how to display Google map on android devices and finally about classes and functions used for location services. “The recent convergence of communication and information technologies has created possibilities unthinkable only a few years ago” Venkatesh(1998). Mobile phones, email, SMS (Short Message Service) and Instant Messenger are new communication technologies, which all contribute to the “death of distance” Cairncross (2001). Instant Messenger is a proprietary, simplified version of Internet Relay Chat, which allows two or more people to carry on a conversation, in realtime, using text based messages with context awareness. Instant Messenger is used to avoid boredom, to socialize, Lai et. al. (2002) and to maintain contact with casual acquaintances, Lee et. al. (2002). Leung (2001) found seven motives for messenger use among college students: affection, inclusion, sociability, entertainment, relaxation, escape and fashion. Mathieson (1991) found people use these mediums to sustain a sense of connection. A lot of portals are available which provide messengers free of charge. Since they are free of charge, they are the preferred services by millions of people around the world. Some of the Mobile Messaging Applications those are generally used are: i.Hike ii.Chat On iii.WhatsApp iv.E-buddy messenger v.Facebook Messenger vi.G Talk vii.Go SMS Pro viii.We-chat

## **2) A cryptographic approach for secure client-server chat application using public key infrastructure**

**Year:** 2016

**Publisher:** ICITST

Encryption and identity authentication have been added to a simple chat application to let clients talk to each other instantly over a secure channel.

A simple chat application has led clients to talk to each other safely by implementing a lot of methods which are important for network security and security concepts

Nowadays, while the popularity of chat applications increases, this popularity brings some security problems with it. A variety of authentication mechanisms and encrypting methods are suggested and applied between server and client to minimize the increasing security problems in literature and marketing companies. In this study, various security measures have been considered for instant messaging applications, a java based client-server chat application developed by Professor Dan Boneh and his assistants from Stanford University has been made secure and a secured chat application model which has three steps has been developed. At the first step, server has been identified itself to certificate authority and password authentication procedure has been performed to identify client itself to server. The second step is called as connection and client connects to chat room via ticket granting ticket (TGT) request in this step. Messages written by clients are sent to server cryptically through symmetric encryption method Advanced Encryption Standard (AES) in the third step which is called as encrypted messaging too. Source code of this application is accessible to everyone from the reference .

### **3) A chat application in Lift**

**Year:** 2010

**Publisher:** IEEE

The application provides a single chat server that takes chat messages and redistributes the messages out to all listeners. Implementation uses a single HTTP connection to poll for changes to an arbitrary number of components on the page. Each component has a version number. The long poll includes the version number and the component's globally unique identifier (GUID). On the server side, a listener is attached to all the GUIDs listed in the long-poll requests. If any component has a higher version number, or the version



number increases during the long-poll period, the listener sends the deltas. Lift's Comet implementation uses a single HTTP connection to poll for changes to an arbitrary number of components on the page. Each component has a version number. The long poll includes the version number and the component's globally unique identifier (GUID). On the server side, a listener is attached to all the GUIDs listed in the longpoll requests. If any component has a higher version number, or the version number increases during the long-poll period, the listener sends the deltas — a JavaScript set describing the change from each version — to the client. The browser applies the deltas and sets the client's version number to the change set's highest version number. Lift integrates long polling with session management so that, if a second request comes into the same URL during a long poll, the long poll is terminated to avoid connection starvation (some browsers have a default maximum of two HTTP connections per named server). Lift also supports Domain Name System (DNS) wild-carded servers for longpoll requests such that each tab in the browser can do long polling against a different DNS server. This avoids connection starvation issues. Lift dynamically detects the container in which the servlet is running. On Jetty versions 6 and 7 and Glassfish version 3.0, Lift uses the platform's continuations implementation to avoid using a thread during the long poll. Lift's JavaScript can sit on top of the jQuery and YUI JavaScript frameworks. The actual polling code includes backoff on connection failures and other graceful ways of dealing with transient connection failures

## 4. REQUIREMENT ANALYSIS

The platform used to implement the Chat Application is identified as Amazon Web Services(AWS).

For Network Connections, a WebSocket API is built using Lambda.

4 Hosts/ Clients are considered but the number can be increased depending on the requirement.

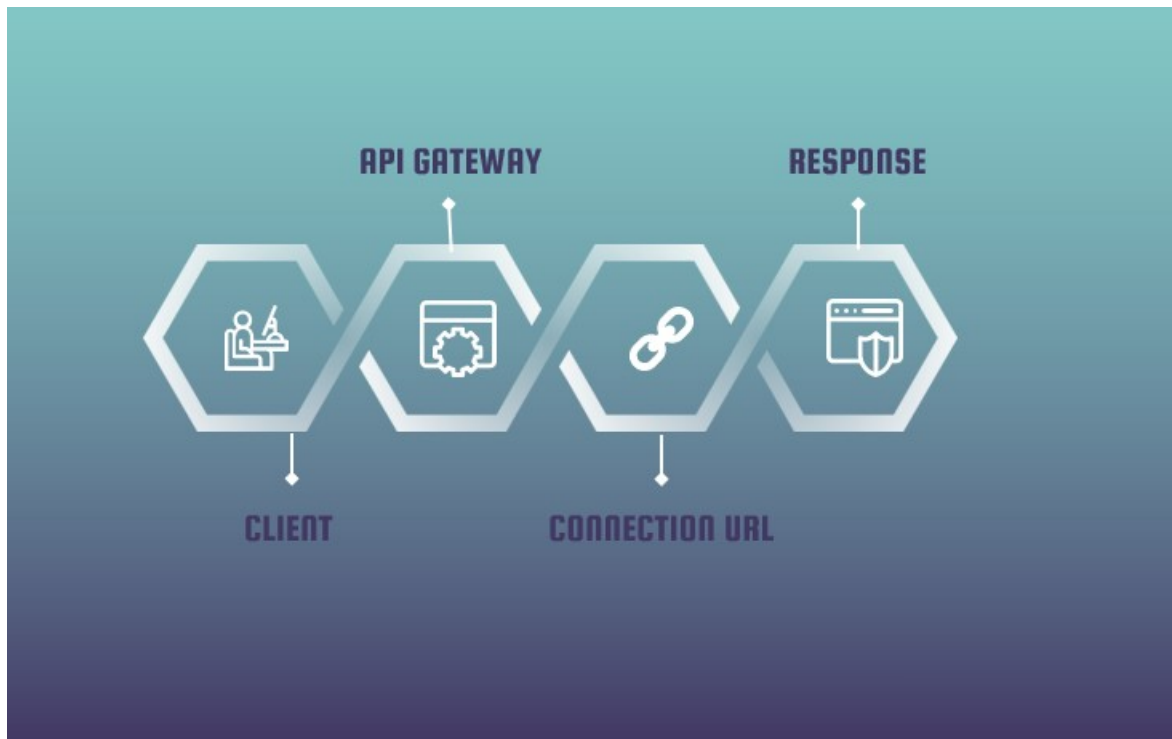
AWS Lambda is identified to be used as Lambda precisely manages the scalability of the functions.

AWS Cloud is used for un-restricted capacity.

Specifications:

- AWS
- Lambda
- 4 or more hosts
- WebSocket API

## 5. THE ARCHITECTURE



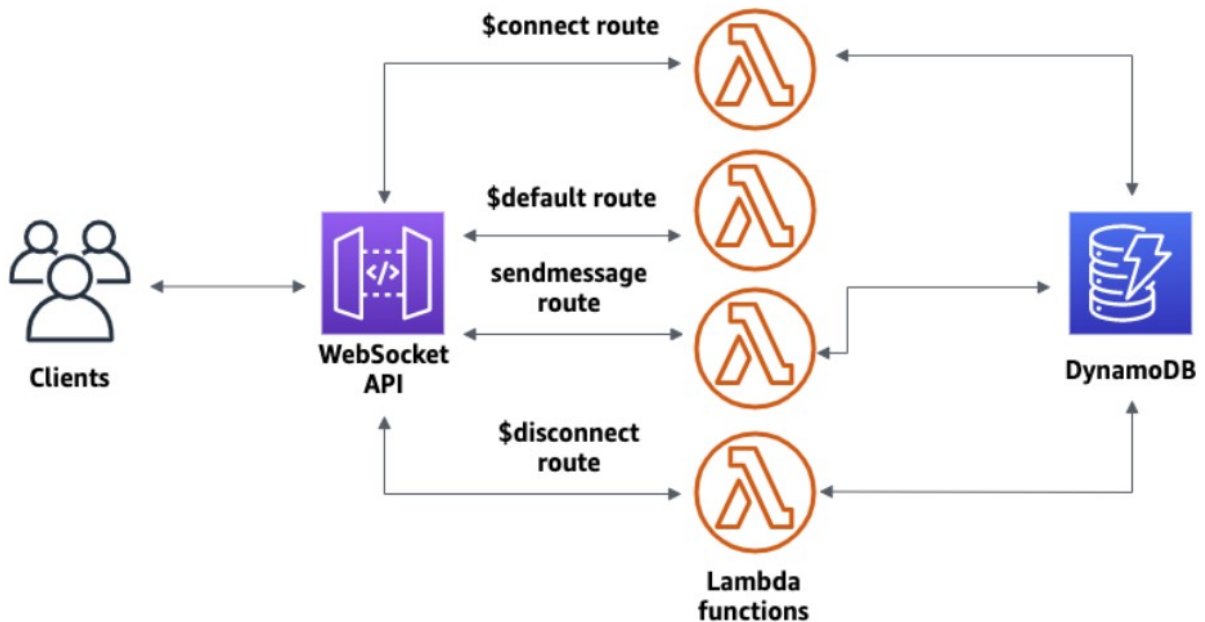
***FIG. 5.1: ARCHITECTURE DIAGRAM***

Figure 5.1 shows:

1. The client sends a message request to the server, in our case, it is a Web-Socket that enables bidirectional communication without needing an actual server. An open connection request is sent from the client side to the socket.
2. An API Gateway with routes is setup to cater to incoming client requests. Client routes may include, setting user name, sending private or public messages, connecting or disconnecting from the socket.
3. Web-Socket URL on AWS to establish a serverless connection. It is responsible for getting response ready that is to be sent to the client.
4. Response is sent to client securely and displayed by means of states and

hooks (ReactJs). It enables a user-friendly experience.

MODULES EXPLAINED:



*Fig 5.1*

Figure. 5.1 shows

### **Routes:**

There are six different routes that have been employed in this project listed as follows:

- Connect
- Disconnect
- Default
- sendPublic
- sendPrivate
- setName

Chat routing feature helps you to prioritize and route incoming chats to the right operators based on rules and conditions that are easy to set up. A WebSocket API to handle client connections and route requests to the Lambda functions. In this project, for simplicity we have set up only six routes that will redirect users to their requested function. With the `sendPublic` route, user can send a message to all the users of the chatroom at once. With the `sendPrivate` route, the user can send a private message to any specific user. The other routes, i.e, `connect`, `disconnect`, `setName` allow users to connect, disconnect and set their user-name on the server respectively. The sixth route called `default` is used to handle any other actions or events that take place on the front end.

## **Backend:**

**Lambda function and DynamoDB:** AWS Lambda is a serverless compute service that runs your code in response to events and automatically manages the underlying compute resources for you. These events may include changes in state or an update, such as a user placing an item in a shopping cart on an ecommerce website. You can use AWS Lambda to extend other AWS services with custom logic, or create your own backend services that operate at AWS scale, performance, and security. AWS Lambda automatically runs code in response to multiple events, such as HTTP requests via Amazon API Gateway, modifications to objects in Amazon Simple Storage Service (Amazon S3) buckets, table updates in DynamoDB, and state transitions in AWS Step Functions. The app creation template from AWS cloud formation is used to create an Amazon DynamoDB table to store the app's client IDs. Each connected client has a unique ID which we will use as the table's partition key. This template also creates Lambda functions that

update your client connections in DynamoDB and handle sending messages to connected clients. We use one single lambda function for all our routes in this application. This lambda function is written in Javascript. It uses event handlers to route the client.

Amazon DynamoDB is a NoSQL database that supports key-value and document data models, and enables developers to build modern, serverless applications that can start small and scale globally to support petabytes of data and tens of millions of read and write requests per second. DynamoDB is designed to run high-performance, internet-scale applications that would overburden traditional relational databases. Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. It's a fully managed, multiregion, multimaster, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications.

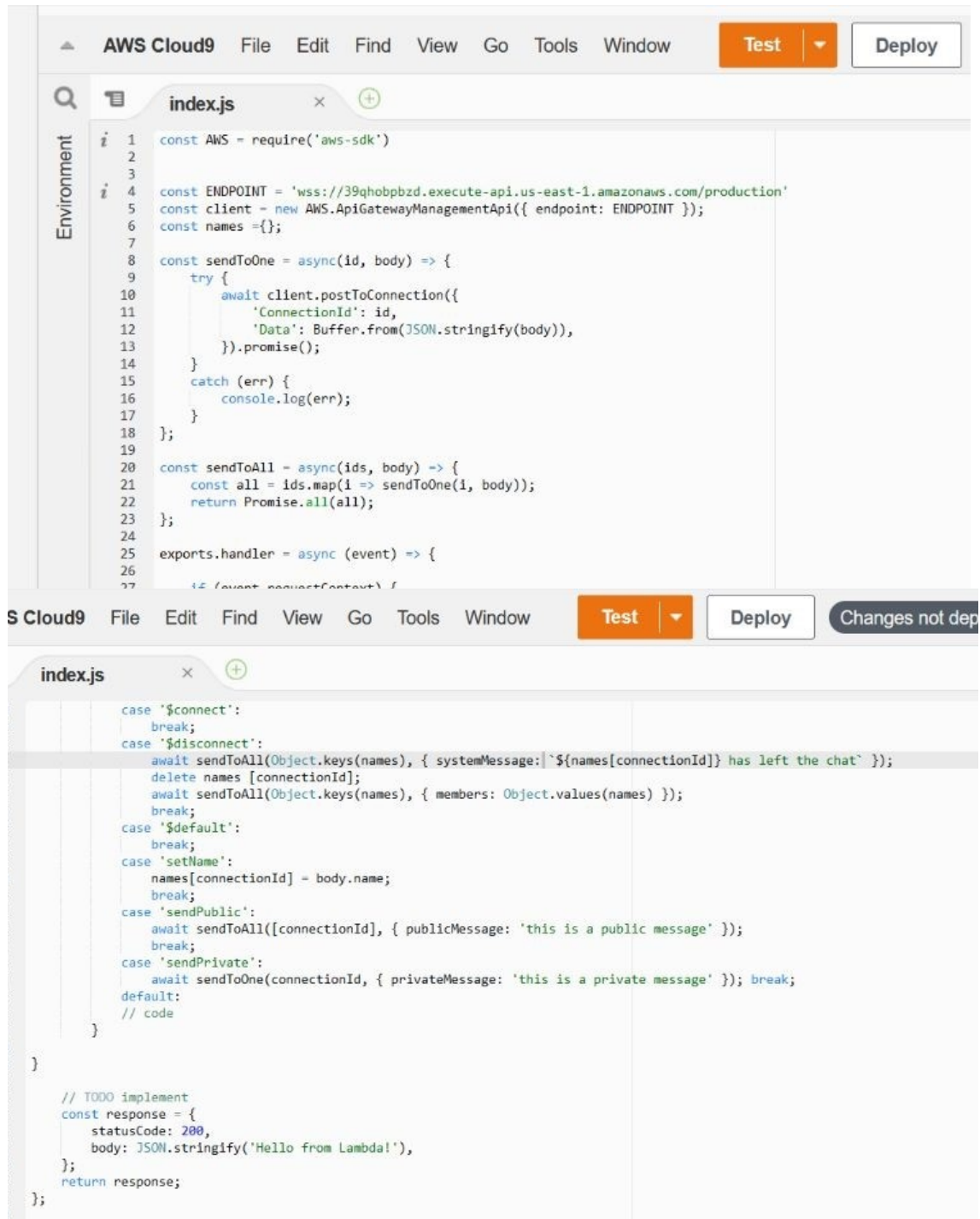
### **Chat UI/UX:**

The front-end of the application is built using ReactJs (states and hooks). React makes it painless to create interactive UIs. It is a component-based library that lets you build high-quality user interfaces for web apps. Along with React as the framework, this project employs Typescript for collecting user data such as user name, text messages and other executable commands. The basic format of the UI includes a list of users displayed on the screen alongside their respective messages after this data is fetched from the Websocket API. The frontend code is majorly written in Javascript/Typescript. TypeScript stands in an unusual relationship to JavaScript. TypeScript offers all of JavaScript's features, and an additional layer on top of these: TypeScript's type system. This means that your existing working JavaScript code is also TypeScript code. The main benefit of TypeScript is that it can highlight unexpected behavior in your code, lowering the chance of bugs.

## 6. IMPLEMENTATION

A small display of the coding implementation of the project:

Lambda Function:



```
const AWS = require('aws-sdk')

const ENDPOINT = 'wss://39qhobpbzd.execute-api.us-east-1.amazonaws.com/production'
const client = new AWS.ApiGatewayManagementApi({ endpoint: ENDPOINT });
const names = {};

const sendToOne = async(id, body) => {
  try {
    await client.postToConnection({
      'ConnectionId': id,
      'Data': Buffer.from(JSON.stringify(body)),
    }).promise();
  } catch (err) {
    console.log(err);
  }
};

const sendToAll = async(ids, body) => {
  const all = ids.map(i => sendToOne(i, body));
  return Promise.all(all);
};

exports.handler = async (event) => {
  if (event.requestContext) {
    case '$connect':
      break;
    case '$disconnect':
      await sendToAll(Object.keys(names), { systemMessage: `${names[connectionId]} has left the chat` });
      delete names [connectionId];
      await sendToAll(Object.keys(names), { members: Object.values(names) });
      break;
    case '$default':
      break;
    case 'setName':
      names[connectionId] = body.name;
      break;
    case 'sendPublic':
      await sendToAll([connectionId], { publicMessage: 'this is a public message' });
      break;
    case 'sendPrivate':
      await sendToOne(connectionId, { privateMessage: 'this is a private message' }); break;
    default:
      // code
  }

  // TODO implement
  const response = {
    statusCode: 200,
    body: JSON.stringify('Hello from Lambda!'),
  };
  return response;
};
```

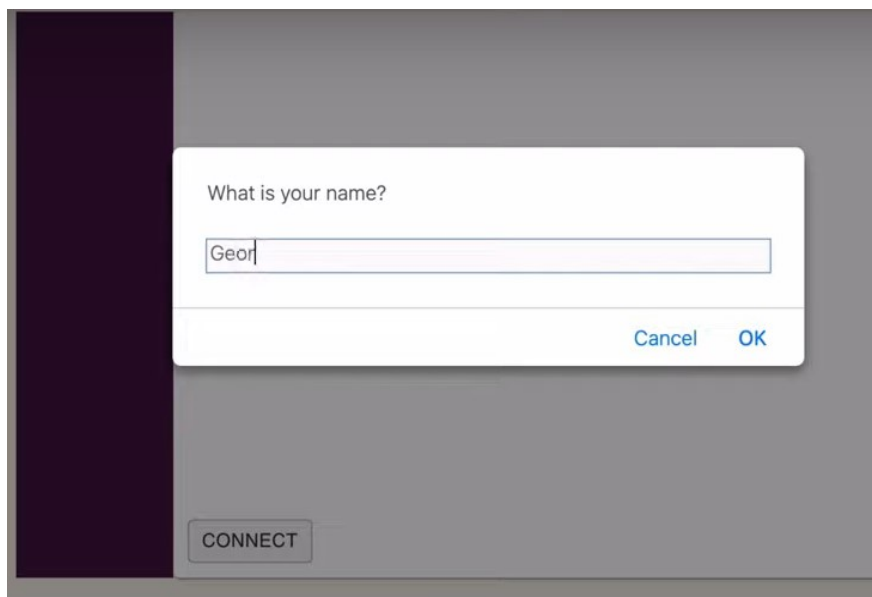
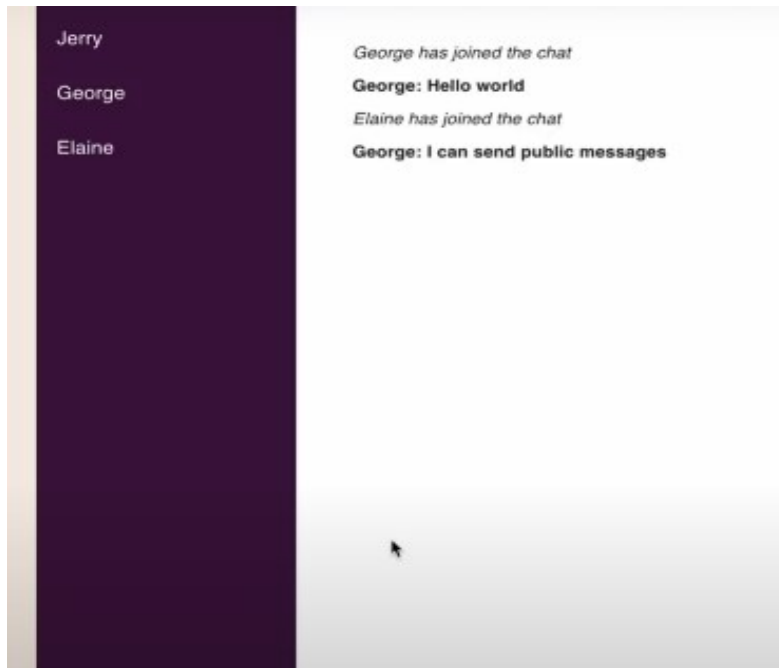
*Fig 6.1*

Figure 6.1 shows the front-end code of the proposed Chat Application written in JavaScript. JavaScript makes the web-page dynamic . It is what powers its interactivity.

JavaScript is a very powerful tool that can do many things for a website. For one, it powers the site's general interactivity. JavaScript makes it possible to build rich UI components such as image sliders, pop-ups, site navigation mega menus, form validations, tabs, accordions, and much more



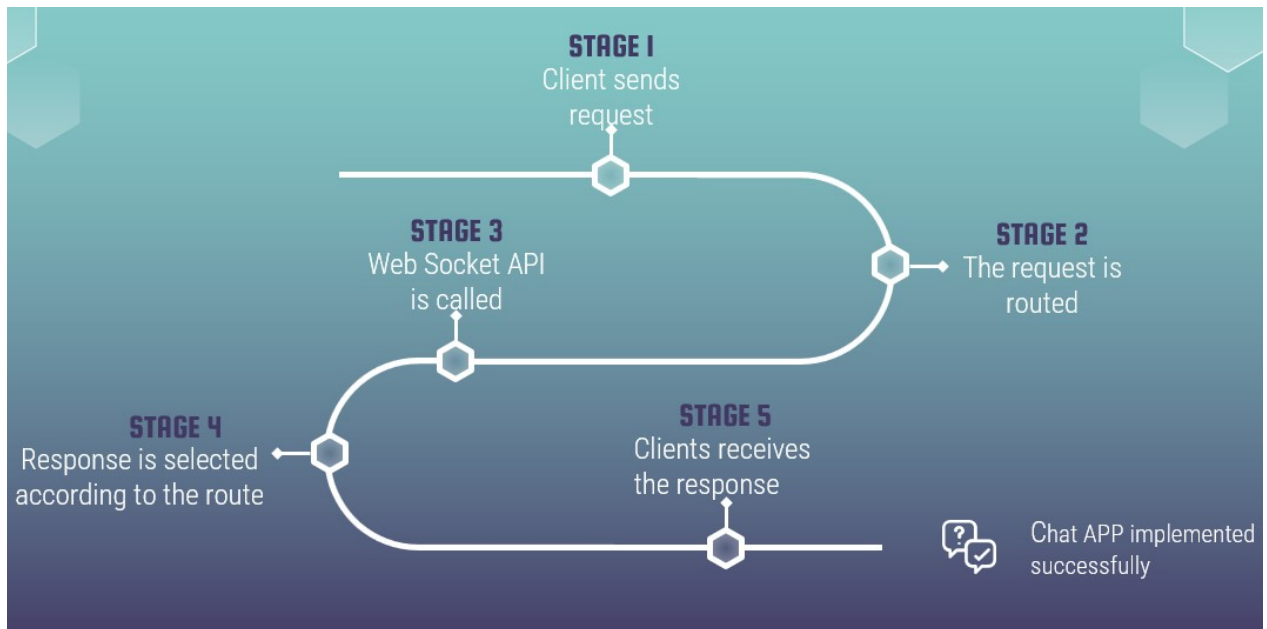
## 7. OUTPUT



*FIG. 7.1: OUTPUT*

Figure 7.1 is a snapshot of what the Chat Application Looks like in action. Its shows

the common room where everyone can read the messages and a host window from where the Host can send a message to other hosts.



*FIG. 7.2: WORKING OF THE MODEL*

Figure 7.2 shows the final output that a user will view which was generated after using AWS lambda function which enables the fetching of data from Websocket API. The web-app allows users to connect to the socket. They can set their user-name or update it and begin communicating. Multiple users can join one chatroom. After the client makes a request, it is routed to the server which fetches the response and that data is parsed and displayed on the screen as shown above using React state and hooks.

## **8. THE CONCLUSION**

AWS lambda and DynamoDB allow us to focus on the code and not the underlying infrastructure which means we are able to build a product **quickly**. With Websocket API being one of AWS's Serverless offerings, the total cost of ownership (TCO) is reduced as there is no need to invest in the maintenance of the underlying infrastructure. Some limitations of the Application are Data Privacy Breach and Low Reliability. Merits include Cost efficiency, increased Security, manageability, scalability and speed.

Thus, a scalable, secure and serverless chat application was implemented.

## **REFERENCES**

Church, Karen, and Rodrigo De Oliveira. "What's up with WhatsApp? Comparing mobile instant messaging behaviors with traditional SMS." *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services*. 2013.

Patinge, Sushant A., and Pravin D. Soni. "A survey on instant message and location sharing system for android." *International Journal of Application or Innovation in Engineering & Management (IJAIEM)* 2.10 (2013): 219-221.

Peng, Bin, Jinming Yue, and Chen Tianzhou. "The Android Application Development College Challenge." *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*. IEEE, 2012.

Skog, Berit. "16 Mobiles and the Norwegian teen: identity, gender and class." *Perpetual contact: Mobile communication, private talk, public performance* (2002): 255.

Shrestha, Ramesh, and Yao Aihong. "Design of secure location and message sharing system for android platform." *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*. Vol. 1. IEEE, 2012.