Simple Chat App using NodeJS and MongoDB Atlas

Project Report Submitted to National Skill Training Institute for Women, Trivandrum in the Partial Fulfillment and Requirements for IBM Advanced Diploma(Vocational) in IT, Networking and Cloud

By

SHEHINA S ADIT/TVM/19/013

ADIT (2019-2021)

Under the supervision of

Mr. Poovaragavan Velumani (Master Trainer, Edunet Foundation)

GOVERNMENT OF INDIA

MINISTRY OF SKILL DEVELOPMENT & ENTREPRENEURSHIP

DIRECTORATE GENERAL OF TRAINING

January 2021

ABSTRACT

To create a simple chat application using Chat App using NodeJS and MongoDB Atlas as its database. Chat apps are valuable for many applications, from interacting on social media to providing customer service. This chat App is a simple application where users can send and can receive messages at same time on two different PC's, and can chat with each other. Using these free chat app it's very easy to continue the conversation in PC .And the messages send and user's name are stored in MongoDB Atlas database.

CONTENTS

ABSTRACT

1.INTRODUCTION

1.1 Objective/ Project Overview

1.2 Introduction About Domain

1.3 Project Description

1.4 Scope of Work

2. SYSTEM ANALYSIS

2.1 Existing System

2.2 Proposed System

3. SOFTWARE DEVELOPMENT ENVIRONMENT

4. SYSTEM DESIGN

4.1 ER Diagrams

4.2 Flow Chart

5. SYSTEM REQUIREMENTS

5.1 Software specification

5.2 Hardware specification

6. COMMAND PROMPT

7. MONGODB ATLAS DATABASE

8. SOURCE CODE

9. SCREENSHOT

10. CONCLUSION

11.REFERENCE

1. INTRODUCTION

The objective of this project is for Building a Real-time Chat App With NodeJS, Express, Socket.IO, and MongoDB . This chat application will provide user's to send and receive messages, and the user name and the messages they send will be stored in MongoDB Atlas database. . This Chat App can be accessed by different types of users. The username and messages stored in database are managed by **Admin**

Admin will have to authority to view all the username and messages. Admin will also be responsible to provide user id and password for accessing to the database. Admin will have the authority to delete the stored information. Users will have the privileges for sending and recieving messages.

1.3 PROJECT DESCRIPTION

To build a simple chatroom that users can use to communicate with other connected users. Any number of users can connect to the chatroom and the messages one user sends become instantly visible to all the users connected to the chatroom.

Our simple chatroom is going to have the following set of features.

- Change the username of the user
- Send messages
- Show if another user is currently typing a message

to build a simple chatroom that users can use to communicate with other connected users. Any number of users can connect to the chatroom and the messages one user sends become instantly visible to all the users connected to the chatroom.

Our simple chatroom is going to have the following set of features.

- Change the username of the user
- Send messages
- Show if another user is currently typing a message

1.4 SCOPE OF WORK

Thinking of a messaging application where users can send messages in real-time. These messages should appear on the other users' application as soon as the messages are sent. If we implement this application like a normal web application, where only the client can initiate requests to the server to receive data, the user has to either refresh the web page regularly to see the newest messages or the client-side should send AJAX requests to the server in short time intervals to retrieve the newest messages. The former of the two is not very user friendly and the latter is a waste of application resources. Then, clearly, we must have a different method to build real-time applications that makes better sense.

2. SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

At present, more than three billion people are using Smartphones with numerous chat apps or messenger apps installed on it. And it consists of different features also.

2.2 PROPOSED SYSTEM

We are using the express, ejs, socket.io, and nodemon packages to build the application.

- Ejs is a popular JS template engine
- socket.io
- Nodemon is a package that restarts the server every time we make a change to the application code. It eliminates the need to manually stop and start the server every time we make a change. Unlike the other packages, will install nodemon as a development dependency since we use it only for development purposes.

3. SOFTWARE DEVELOPMENT ENVIRONMENT

In a world where the value of time is steadily increasing, building applications that users can interact with in real-time has become a norm for most of the developers. Most of the applications we see today, whether they are mobile, desktop, or web applications, have at least a single real-time feature included. As an example,

real-time messaging and notifications are two of the most commonly used real-time features used in applications.

In this project, I am introducing you to the development of real-time applications using Node.js by building a real-time chat. In fact, Node is one of the best programming languages out there to build real-time applications due to its event-driven and asynchronous nature. Before diving into building a real-time application head-first, we'll see what kind of real-time applications we can build using Node.js.

For building real time Applications WebSocket provides the solution needed. WebSocket is a communication protocol that allows both the client and server to initiate communication. In other words, with WebSocket, the server can send data to the client any time without the client having to request data first. In the case of the previous messaging application, WebSockets can be used to instantly send messages to all the users through the server. Also WebSocket API can be used to communicate using WebSockets when building applications.

Socket.io

However, when implementing a real-time application using Node, don't have to directly use the WebSocket API. Instead, Javascript and Node.js library Socket.io, which is an API to the WebSocket API, provides a much simpler implementation of WebSockets for us to use. In this tutorial, we will be using Socket.io to create and manage WebSocket connections between the client and the server. This is a simple chatroom were users can use to communicate with other connected users. Any number of users can connect to the chatroom and the messages one user sends become instantly visible to all the users connected to the chatroom.

Install dependencies Used are express, ejs, socket.io, and nodemon packages to build the application.

The next feature we are going to implement is sending messages. Here things start

to get a little bit different, so far we said that every time the front-end emits a message the server will receive it, however in our new case, the front-end needs to emit a new_message event, which then will need to be sent to all the connected clients, so that they can print the new message.

First, will set up the front-end to emit a new_message event when a new message is submitted. Since the client-side should also be configured to receive new messages other users send from the server, the application should also listen to receive_message events on the front-end and show the new message on the web page appropriately.

MongoDB Atlas is a Fully Managed Database Service which takes care of time-consuming and costly administration tasks so we can get the database resources we need, when we need them.

MongoDB Atlas is a cloud-based NoSQL database service developed by MongoDB Inc. It was developed to offer a flexible, scalable, and on-demand platform to eliminate the need for costly infrastructure, configurations, and maintenance.

MongoDB Atlas provides all the features of MongoDB—without the need to worry about database administration tasks such as:

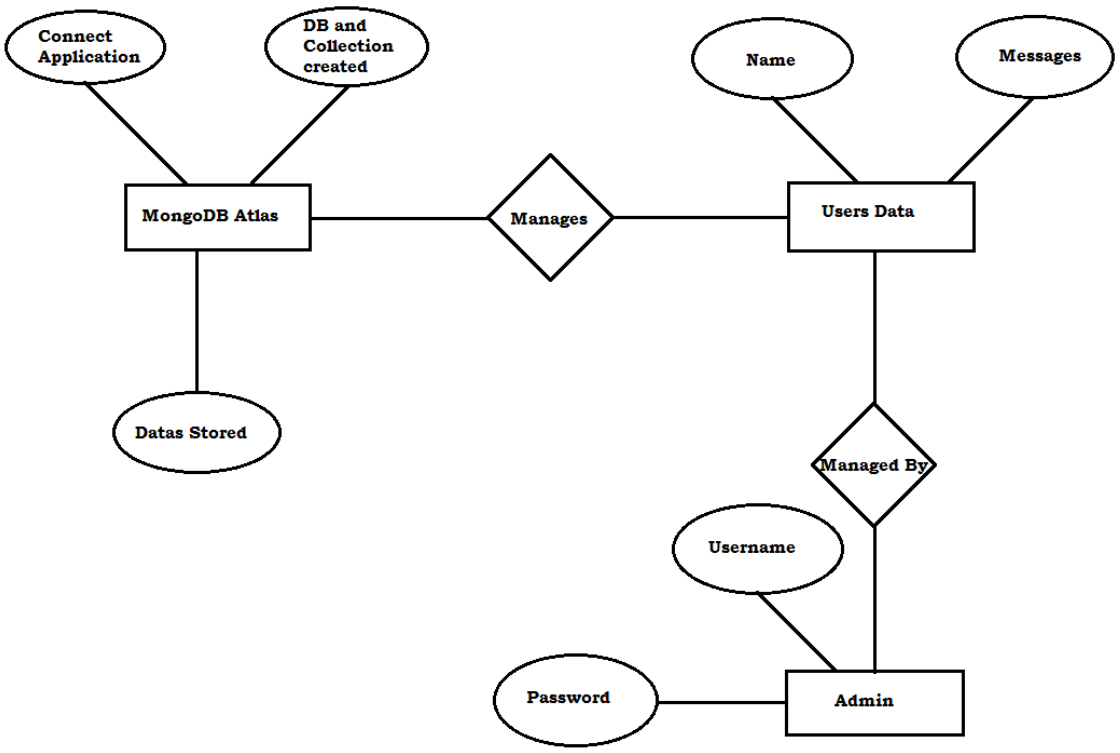
- Infrastructure provisioning
- Database configurations
- Patches
- Scaling
- Backups

The datas which are being entered in chat application is stored on MongoDB Atlas Database .

4. SYSTEM DESIGN

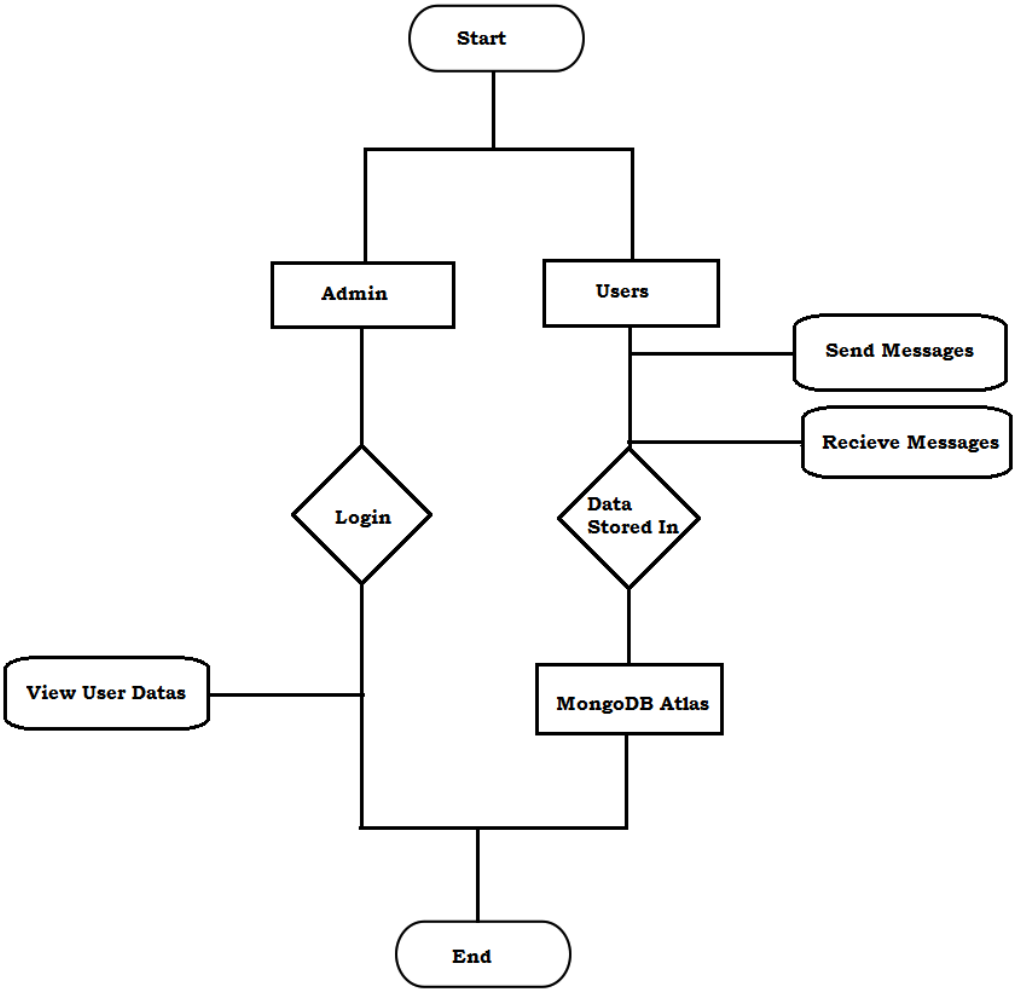
4.1 ER DIAGRAM

An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database.



4.2 FLOW CHART

A flowchart is a diagram that depicts a process, system or computer algorithm. They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-to-understand diagrams. Flowcharts, sometimes spelled as flow charts, use rectangles, ovals, diamonds and potentially numerous other shapes to define the type of step, along with connecting arrows to define flow and sequence.



5. SYSTEM REQUIREMENTS

5.1 SOFTWARE SPECIFICATION

Operating System : Windows 7 or above

Front End : HTML, CSS, Java Script

Back End : MongoDB Atlas, express, ejs, NodeJS, Socket.io

Code Editor : , Visual Code

Browser : Google Chrome

5.2 HARDWARE SPECIFICATION

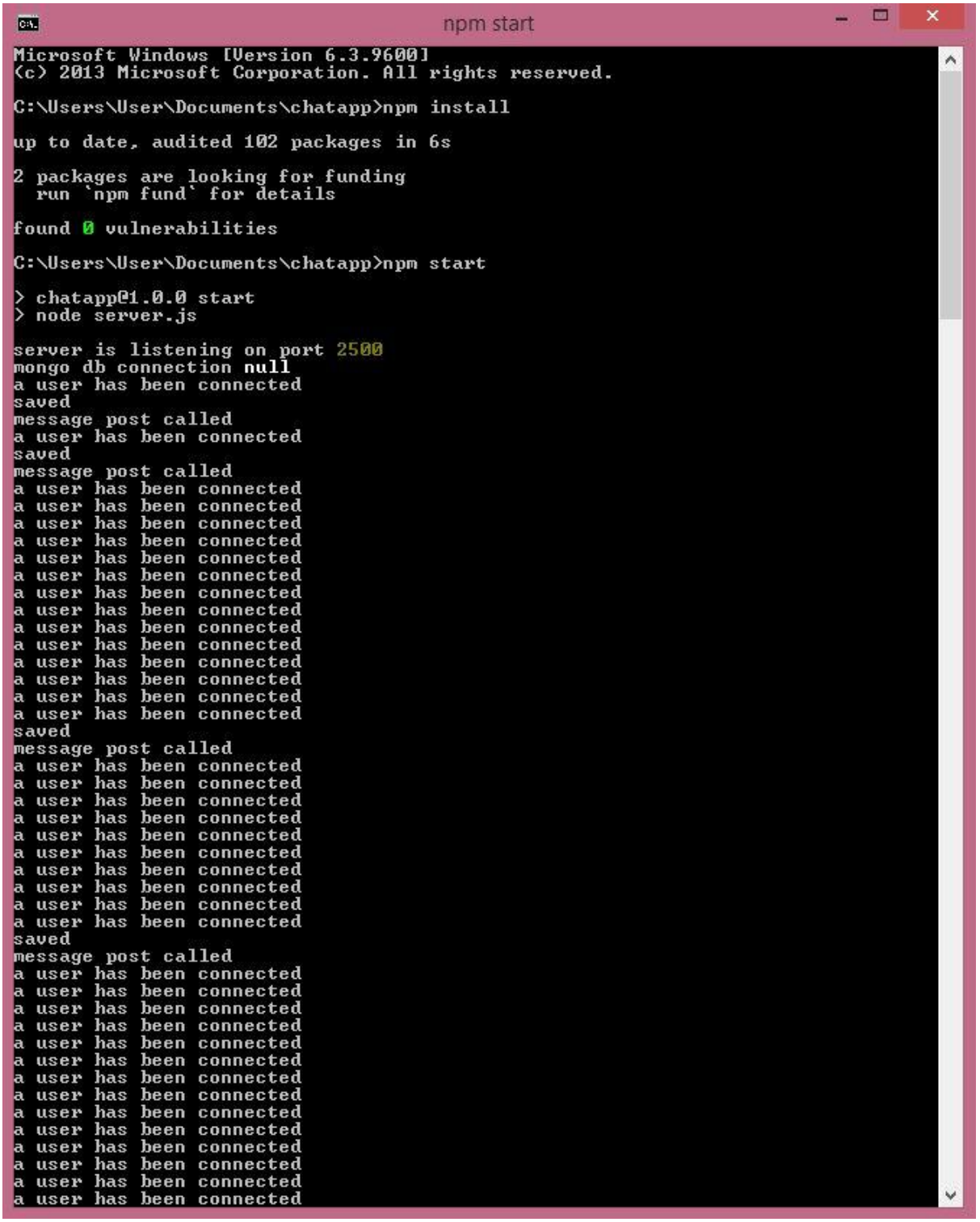
RAM : 1 GB or above

Processor : 1 GHz or more

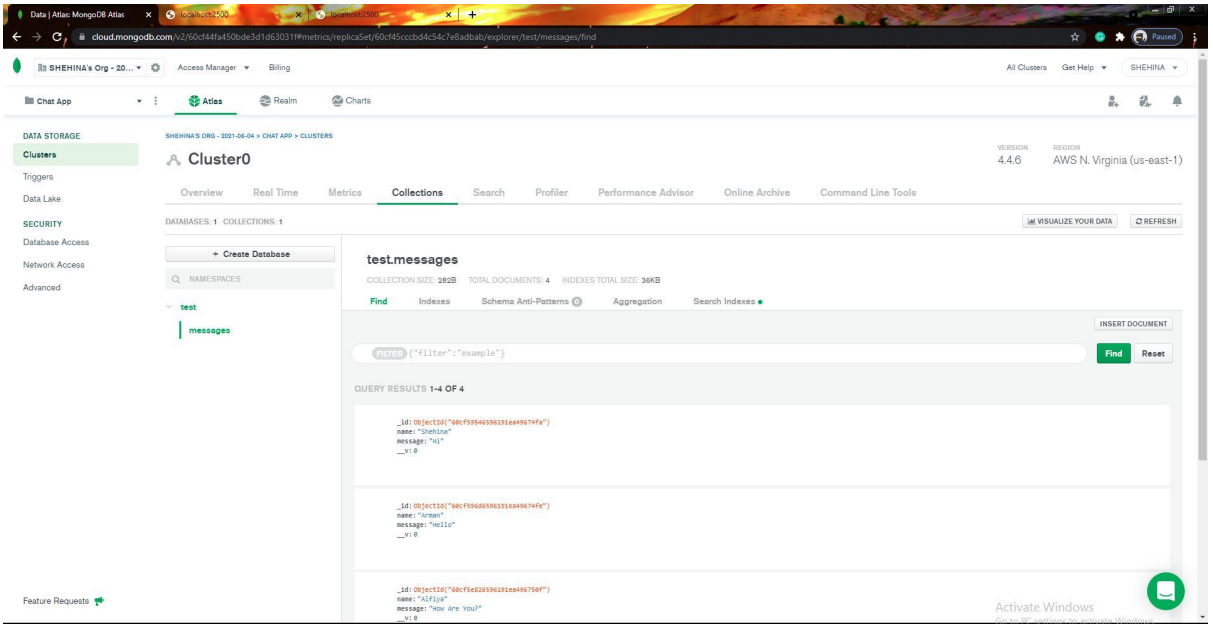
Hard Drive : 32 GB or above

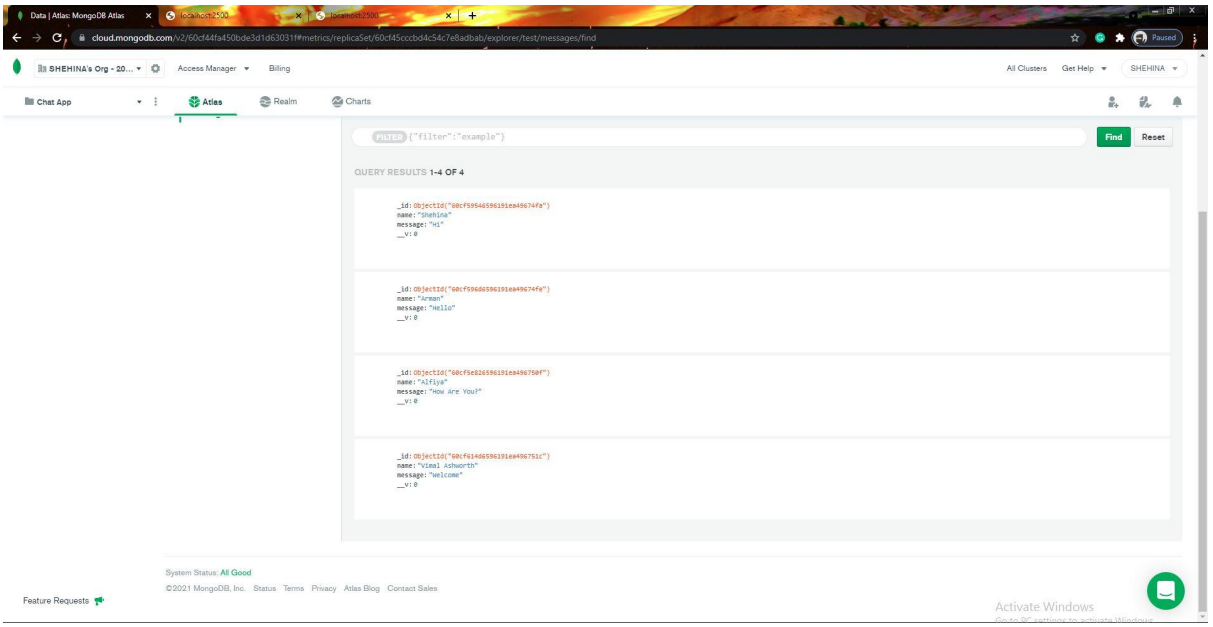
Network Connectivity: LAN or Wi-Fi

6. Command Prompt



7. MongoDB Atlas Database





8. SOURCE CODE

index.html

```
<!DOCTYPE html>

<!--[if lt IE 7]>          <html class="no-js lt-ie9 lt-ie8 lt-ie7">
<![endif]-->

<!--[if IE 7]>         <html class="no-js lt-ie9 lt-ie8"> <![endif]-->

<!--[if IE 8]>         <html class="no-js lt-ie9"> <![endif]-->

<!--[if gt IE 8]><!-->

<html class="no-js">

  <!--<![endif]-->

  <head>

    <meta charset="utf-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <title></title>
```

```
    <meta name="description" content="" />

     <meta name="viewport" content="width=device-width, initial-scale=1"
/>

    <link rel="stylesheet" href="style.css" />

    <link

     rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min
.css"

integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgF
AW/dAiS6JXm"

     crossorigin="anonymous"

   />

    <script

     src="https://code.jquery.com/jquery-3.3.1.min.js"

     crossorigin="anonymous"
```

```
 ></script>

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

</head>

<body>

  <div class="chatbox">

    <div class="chatlogs">

      <div class="chat" id="messages"></div>

    </div>

    <div class="chat-form">

      <h3>Send Message</h3>

      <input id="name" class="form-control" placeholder="name" />

      <br />

      <textarea

       id="message"
```

```
        class="form-control"

        placeholder="message"

      ></textarea>

       <br />

       <button id="send">Send</button>

    </div>

  </div>



<script>

    var socket = io();



    // ========= Start of document ready function =======

    $(() => {

      $("#send").click(() => {
```

```
    var message = {

      name: $("#name").val(),

      message: $("#message").val()

    };

    postMessage(message);

  });

  getMessage();

});



// ========= End of document ready function =========

socket.on("message", addMessage);



function addMessage(message) {

  $("#messages").append(
```

```
                    `<p  class="user-name">  ${message.name}  </p>  <p
class="chat-message"> ${message.message} </p>`

        );

    }



    function getMessage() {

      $.get("http://localhost:2500/messages", data => {

        data.forEach(addMessage);

      });

    }



    function postMessage(message) {

      $.post("http://localhost:2500/messages", message);

    }
```

```
    </script>

    </body>

</html>
```

Style.css

```
body {

  font-family: Arial, Helvetica, sans-serif;

}

* {

  box-sizing: border-box;

}



.chat-form {

  width: 100%;
```

```
  padding: 12px;

  border: 5px solid #ccc;

  border-radius: 4px;

  box-sizing: border-box;

  margin-top: 6px;

  margin-bottom: 16px;

  resize: vertical;

}



#send {

  background-color: #04aa6d;

  color: white;

  padding: 12px 20px;

  border: none;
```

```
  border-radius: 4px;

  cursor: pointer;

}

#send:hover {

    background-color: #45a049;

  }

.chatbox {

  border-radius: 5px;

  background-color: #f2f2f2;

  padding: 20px;

}
```

```css
.chatlogs{

  width: 100%;

  padding: 12px;

  border: 5px solid #ccc;

  border-radius: 4px;

  box-sizing: border-box;

  margin-top: 6px;

  margin-bottom: 16px;

  resize: vertical;

}

#messages{

  background-color: #e5e7eb;

}
```

Server.js

```
var express = require("express");
var app = express();
var http = require("http").Server(app);
var io = require("socket.io")(http);
var bodyParser = require("body-parser");
var mongoose = require("mongoose");
app.use(express.static(__dirname));

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
var dbUrl =
"mongodb+srv://Shehina:shehina1998@cluster0.w1zdm.mongodb.net/test?retr
yWrites=true&w=majority";
var Message = mongoose.model("message", {
  name: String,
  message: String
});
```

```
app.get("/messages", (req, res) => {
  Message.find({}, (err, messages) => {
    res.send(messages);
  });
});


app.post("/messages", async (req, res) => {
  try {
    var message = new Message(req.body);
    var savedMessage = await message.save();
    console.log("saved");


    var censored = await Message.findOne({ message: "badword" });


    if (censored) await Message.remove({ _id: censored.id });
    else io.emit("message", req.body);
    res.sendStatus(200);
  } catch (error) {
    res.sendStatus(500);
    return console.error(error);
  } finally {
    console.log("message post called");
  }
});
```

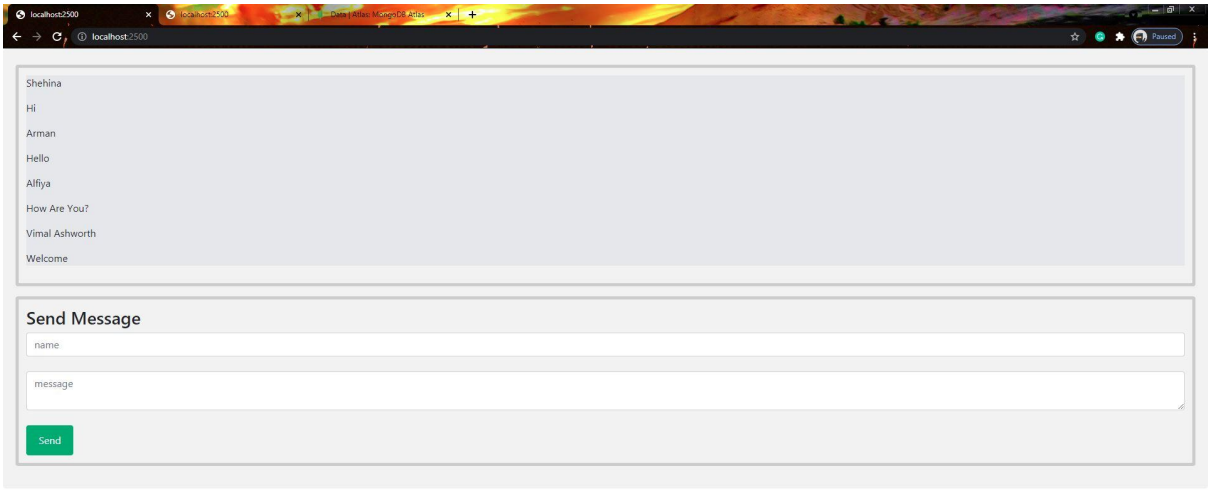
```
io.on("connection", socket => {
  console.log("a user has been connected");
});

mongoose.connect(
  dbUrl,
  { useNewUrlParser: true, useUnifiedTopology: true },
  err => {
    console.log("mongo db connection", err);
  }
);

var server = http.listen(2500, () => {
  console.log("server is listening on port", server.address().port);
});
```

Package.json

```
{
  "name": "chatapp",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
```

```
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "http": "0.0.1-security",
    "mongoose": "^5.12.14",
    "socket.io": "^4.1.2"
  },
  "devDependencies": {},
  "description": ""
}
```

9. SCREENSHOT



10. CONCLUSION

Today, using real-time features with desktop, mobile, and web applications has almost become a necessity. This project includes a real-time chatroom created with the help of using NodeJS and MongoDB Atlas.

11. REFERENCE

1. **https://livecodestream.dev/post/a-starter-guide-to-building-real-time-applications-with-nodejs/**
2. **https://kb.objectrocket.com/mongo-db/simple-chat-app-using-nodejs-and-mongodb-atlas-part-4-891**
3. **https://www.mongodb.com/cloud/atlas**
4. **https://www.bmc.com/blogs/mongodb-atlas/**