

CRN : 592  
Course name : Cloud Computing And Applications  
Instructor : Dr. Phu Phung  
Name : Vivek Manikandan  
UD email : [manikandanv2@udayton.edu](mailto:manikandanv2@udayton.edu)

**TEAM REPOSITORY URL :**

<https://bitbucket.org/cca-vivek-pravin/cca-project-vp/src/sprint3/>

## **SPRINT 3 - Realtime Web Application with authenticated private chat system with chatbot**

### **Introduction:**

The application is mainly to share the private chat messages and the users can ask the fitness questions to the chatbot we have implemented.

We designed this application in such a way that users can simply create an account in our site through the signup page we provide and can start using the application once after they login by giving their authenticated credentials.

The primary goal of this application is to develop a private chat system where the authenticated users only can send/receive the messages and we also have the chat box for every user where it can be used to post the public messages if they wish. Here we are also storing the private messages and reciprocating it to the appropriate authenticated users.

The second main important functionality is the chatbot system which would be used to know the customers fitness status and also we tried the chatbot to give the response for the basic questions along with two important questions those are checking the BMI status and water intake suggestions.

We have created all these functionalities in 3 sprints and are planning to launch it to market once all the testing is done.

**Application URL:** <https://cca-vivekpravin-cloud.azurewebsites.net>

**Authentication Microservice:** <https://cca-sprintauth.herokuapp.com>

**Chatbot Microservice1:**

[https://cca-sprint1-waterintake.azurewebsites.net/water\\_intake?weight=](https://cca-sprint1-waterintake.azurewebsites.net/water_intake?weight=)

**Chatbot Microservice2:**

[https://cca-cloud-microservices.herokuapp.com/bmi\\_calc?height=&weight=](https://cca-cloud-microservices.herokuapp.com/bmi_calc?height=&weight=)

## **Design and Implementation:**

The basic design we planned and implemented is the user can signup and login easily with our front end UI and can start

whenever an authenticated user logged in, the user will get added in the side user list we displayed and once they logout, the list will refresh automatically by our backend program. And users can click on friends from the user list and can start to do private chats. We have also implemented the backend functionality to store and retrieve the private messages to the appropriate users.

The chatbot functionality we have implemented will be available for every user in the first place in the users list through which the users can ask their questions about their fitness and get the answers.

### **Implementation sprint1:**

We created two microservice in which microservice 1 is developed to show the water intake as per the body weight and the algorithm is implemented for that to calculate and show the accurate water level per day.

Input: One input field (Weight)

**Computation:** performs the algorithm by taking the input and generating the output.

**Output:** a string with the output (water intake measured in ounces or gallons).

**Microservice 2** is used to show BMI and fitness status, in which the proper formulas have been used for BMI calculation with respect to weight and height and few conditions are also implemented to show the fitness status as per the BMI.

Input: Two input field (Weight & Height)

**Computation:** performs the algorithm by taking the inputs and generating the output with the status of the fitness.

**Output:** a string with the output (BMI and the fitness is obese or fit)

Later we have used both the above microservices under the chatbot functionality

### **Implementation sprint2:**

We have provided the users with three different screens and we have implemented the functionality that the user would be able to see only one screen at a time which is being used dedicatedly.

The first screen is the signup screen, in which the data entered will be verified and validated in the backend to ensure it is matching our credential requirements and whether the username already exists.

The second screen is the sign in screen, in that the data will be validated from the database to ensure the data is actually present. The third chat screen will be visible only if you pass all three stages, through which the messages will be displayed after verifying if it is from the authenticated users.

We created a Realtime Web Application with an authenticated private chat system in which We have designed two screens after login, one is a side bar which displays userlist and another is an auto popup window to send or receive messages.

The sidebar is implemented to display the user list whoever is connected with the application and whenever a new user is added, the implementation has been given to refresh the list and add the user in that list after it passes the complete authentication written at the backend and when any user is logging off from the application, the user will be removed from that list.

The pop window will appear if you click on an user from the list to start the chatting and whenever you receive the first message from the user, it will popup automatically.

We have written the functionality to show both the message and the sender name to the receiver to uniquely identify. And also for each different user a new popup window will be shown. We also have the chat screen in the page for the authenticated users to chat and it will display it to all the users who are logged in.

### **Implementation sprint 3:**

In this part we have implemented the functionality where when the chats with his friends and those chats will get stored through our backend functionality in mongo db with the appropriate sender and receiver details with the two API's we have given.

API 1: this API will be called whenever the users sends some chats with their friends and those chats will be stored with the appropriate sender and receiver details with the timestamp and with the unique id.

API 2: This will be called whenever the user receives the messages to display the stored chats to the appropriate sender and receiver to bring back the last sent and received messages. We are calling this API also when the users selects the friends to chat from the user list

Chatbot: We have created a chatbot UI to respond to the users queries and we have give both the microservices for the fitness along with this chatbot functionality and when the user asks for the BMI status to the bot by giving the weight and height, it

will redirect to the microservice to get the BMI status. And also the same process has been followed when the user tries to get their water consumption suggestion.

## Demo Screenshots:

### Sprint 1:

The Microservice 1 takes the Input of Weight and gives the response of the Water consumption level per day.

The screenshot displays a web application interface. At the top, there is a clock icon and the text "Current time: Wed Jun 23 2021 17:10:33 GMT-0400 (Eastern Daylight Time)". Below this, the heading "WATER INTAKE BY WEIGHT" is centered. The main form includes a label "weight in kgs:" followed by an input field containing "175" and a green button labeled "Get your per day water intake". Below the button, an orange banner displays the response: "Response from server: you have 175 lbs weight and you have to intake 116.67 ounces or 3.52 litres of water per day".

Below the application view, the Chrome DevTools Network tab is open, showing a list of requests. The selected request is "water\_intake?weight=175". The details pane for this request shows the following information:

- Request URL: `https://cca-vivekpravin-cloud.azurewebsites.net/water_intake?weight=175`
- Request Method: GET
- Status Code: 200 OK
- Remote Address: 20.49.104.34:443
- Referrer Policy: strict-origin-when-cross-origin

The Microservice 2 which sends the height and weight as requested and gets back the BMI and fitness status as response.

The screenshot shows a web browser displaying a web application and its network traffic in the developer tools.

**Web Application:**

- Header:** CPS 592 CLOUD COMPUTING AND APPLICATION, By Phu Phung, by Vivek Manikandan and Arun Pravin
- Clock:** A clock showing the current time.
- Current time:** Wed Jun 23 2021 17:13:57 GMT-0400 (Eastern Daylight Time)
- WATER INTAKE BY WEIGHT:**
  - Input: weight in kgs: 175
  - Button: Get your per day water intake
  - Response from server: you have 175 lbs weight and you have to intake 116.67 ounces or 3.52 litres of water per day
- BMI & FITNESS:**
  - Input: Height in cms: 175
  - Input: Weight in kgs: 65
  - Button: BMI & FITNESS INSTRUCTOR
  - Response from server: Your weight is 65 kg and height is 175 cm and your BMI is 21.22. (The BMI shows that you are fit)

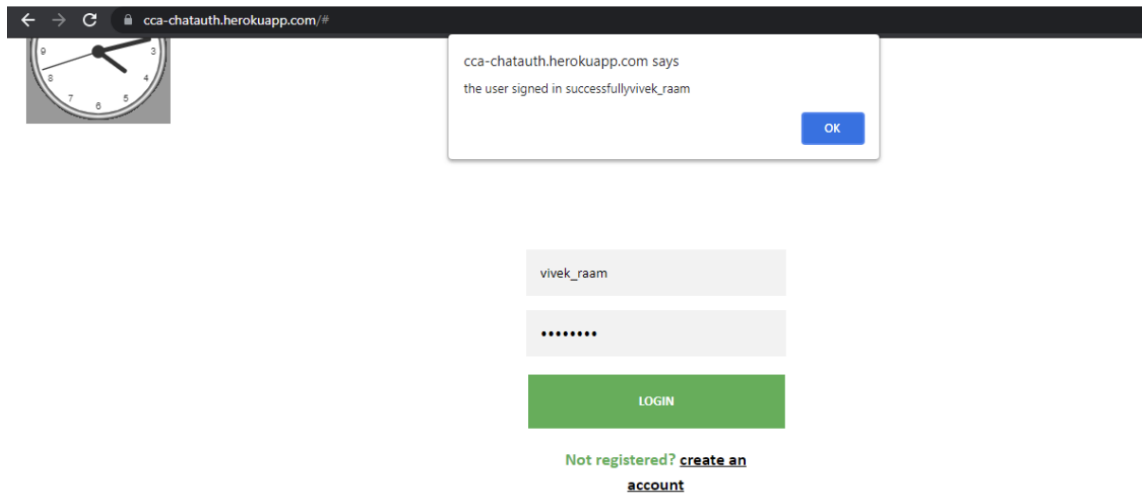
**Network Traffic:**

- Name:** bb-website
- Request URL:** https://cca-cloud-microservices.herokuapp.com/bmi\_calc?height=175&weight=65
- Request Method:** GET
- Status Code:** 200 OK
- Remote Address:** 52.22.80.219:443
- Referrer Policy:** strict-origin-when-cross-origin
- Response Headers:**
  - Access-Control-Allow-Origin: \*
  - Connection: keep-alive
  - Content-Length: 97
  - Content-Type: text/html; charset=utf-8
  - Date: Wed, 23 Jun 2021 21:13:48 GMT
  - Etag: W/"61-c0A14yfgxCfGCUmFys9QpV10qbE"
  - Server: Cowboy
  - Via: 1.1 vegur
  - X-Powered-By: Express

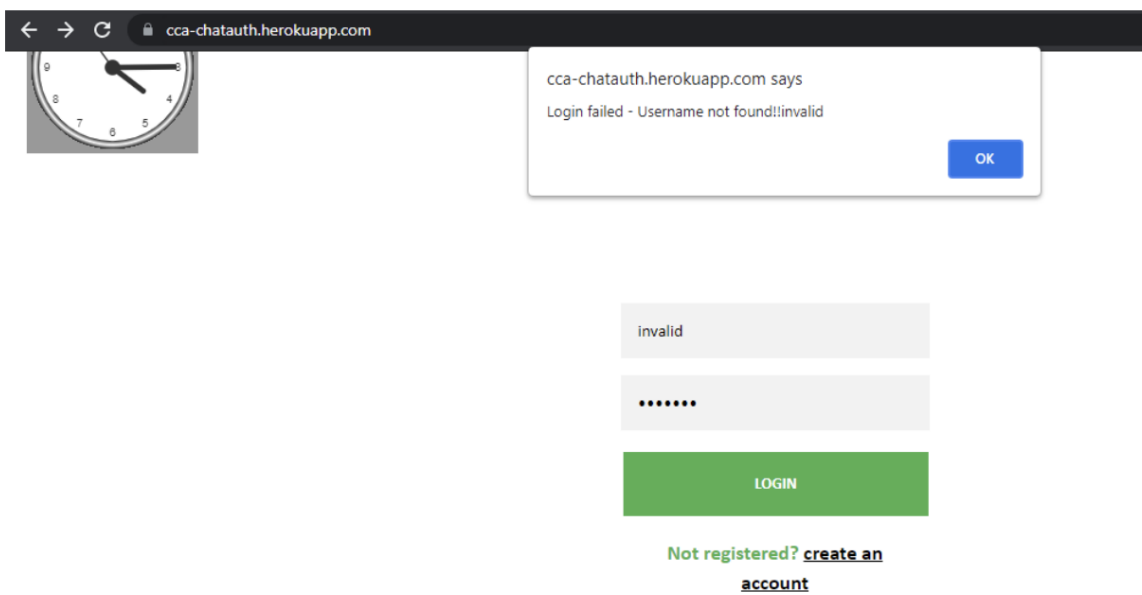
## Sprint 2:

A user can login to system with valid username/password

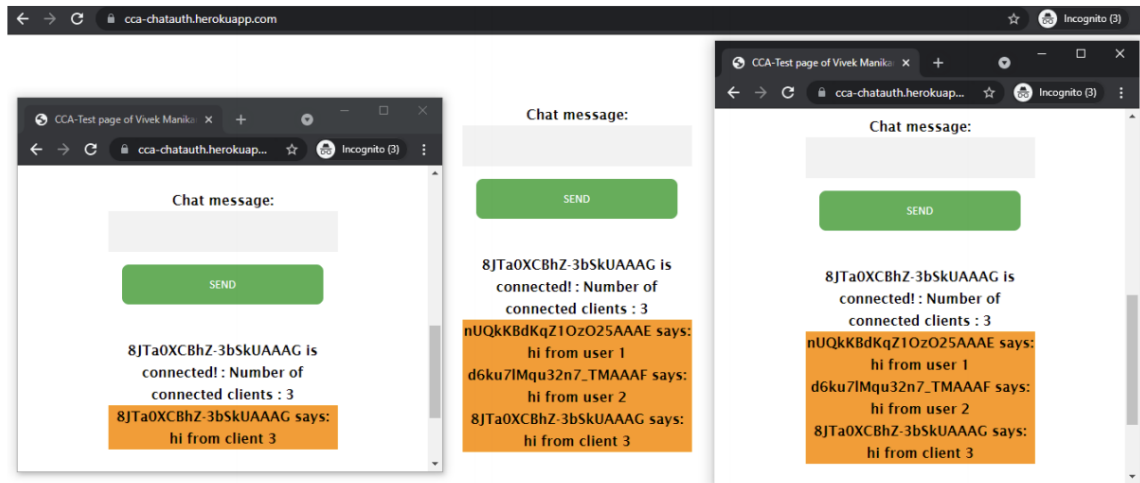
Not registered? [create an account](#)



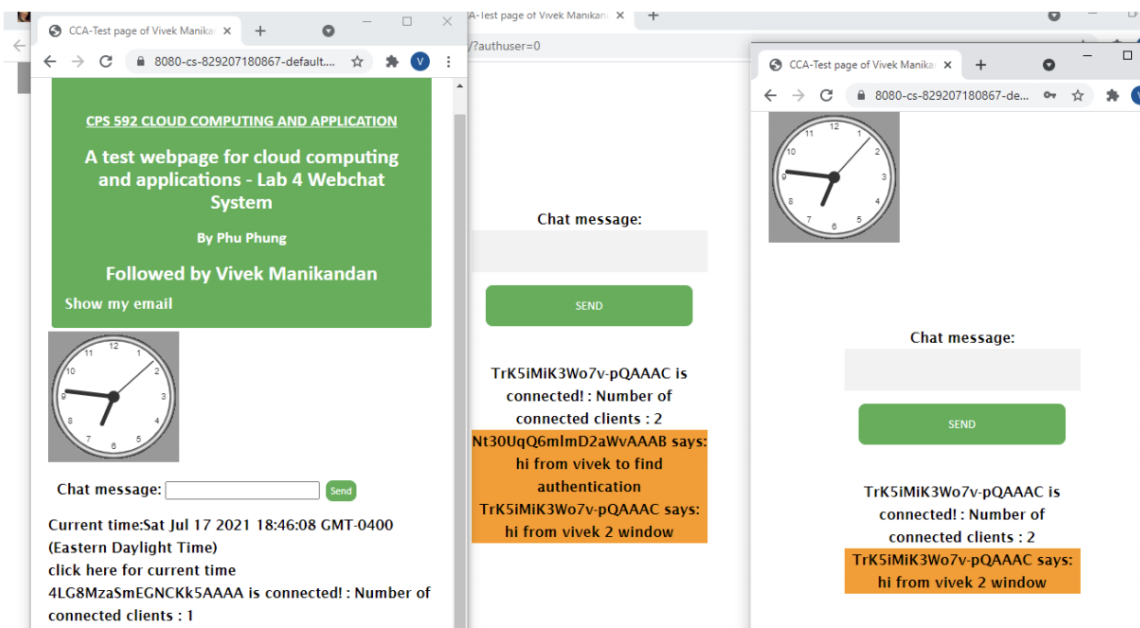
## Invalid username/password



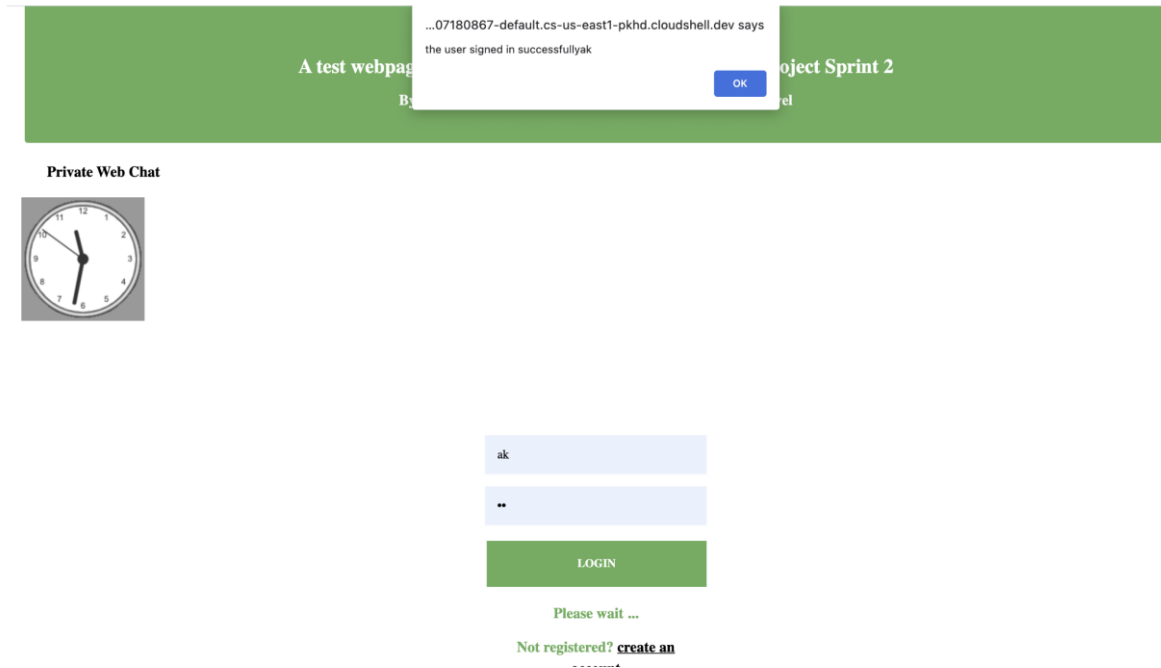
## Chat service - with 3 clients connected authenticated



## Chat server with 3 clients 2 authenticated and 1 unauthenticated



## Login Screen:

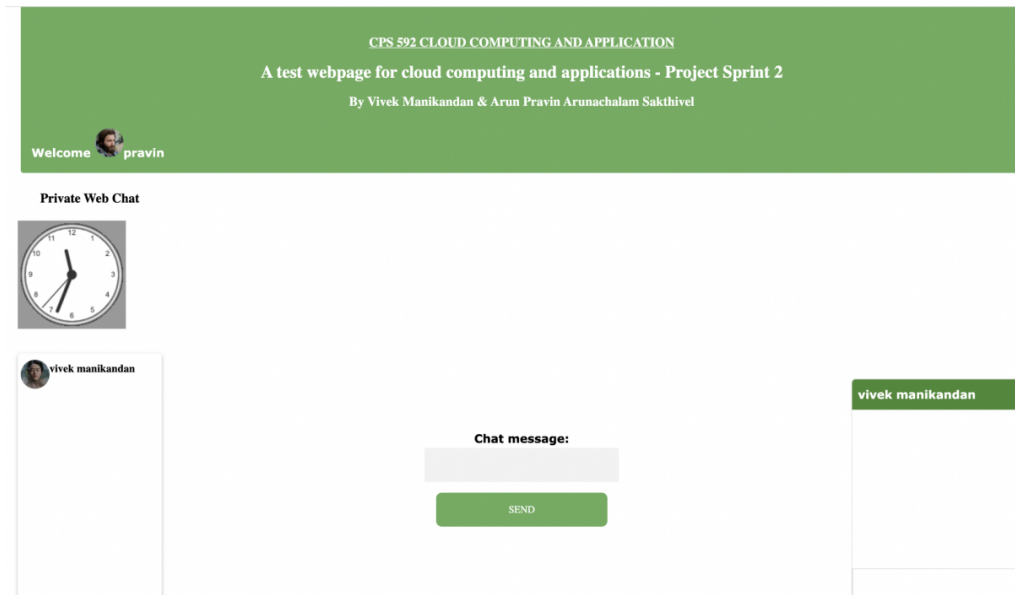


Displays side users lists whoever logged in.

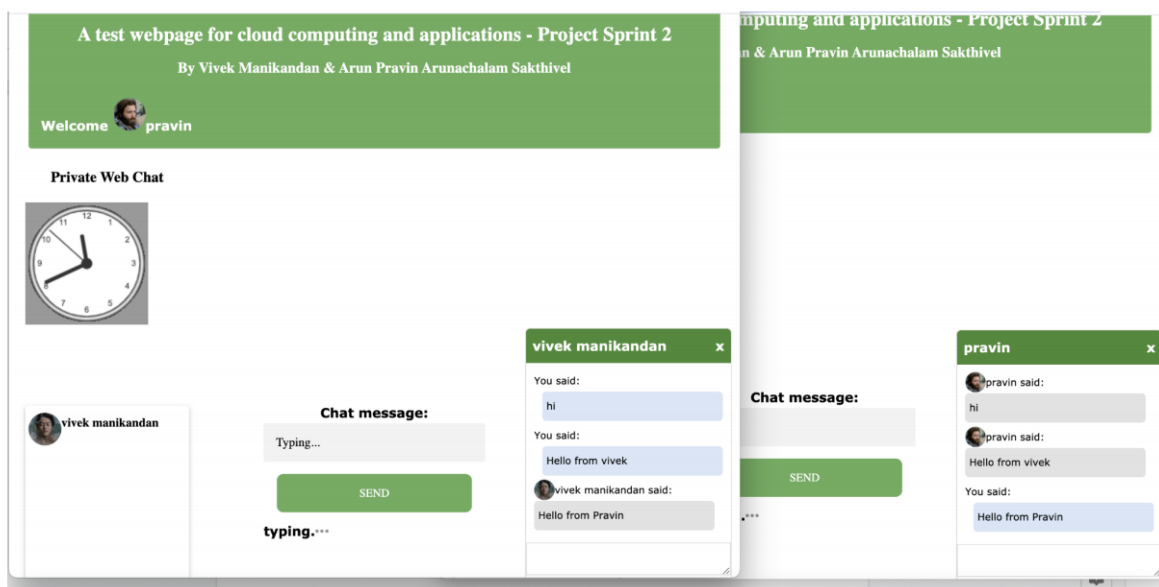




Pop up appears when selecting the friends to chat with.




Private chat with two users.






## Sprint 3:

### Private chat store and load

Welcome  vivek manikandan

Private Web Chat



 chatbot  
 arunpravin

Chat message:

Send

arunpravin is connected! : Number of connected clients : 2

arunpravin

arunpravin said:


how are you

You said:


i am good pravin



You said:

how are you doing?

Welcome  arunpravin

Private Web Chat



 chatbot  
 vivek manikandan

Chat message:


Send

arunpravin is connected! : Number of connected clients : 2


vivek manikandan

You said:

how are you

 vivek manikandan said:

i am good pravin

 vivek manikandan said:





how are you doing?

## DB data:

```
_id: ObjectId("610c06d970b3a2e41e185eb5")
sender: "vivek"
receiver: "arunpravin"
message: "hi"
timestamp: 2021-08-05T15:42:17.953+00:00
```

>

```
_id: ObjectId("610c074f70b3a2e41e185eb6")
sender: "arunpravin"
receiver: "vivek"
message: "how are you"
timestamp: 2021-08-05T15:44:15.221+00:00
```



< PREVIOUS





21-40 of many results

NEXT >

```
_id: ObjectId("610c075a70b3a2e41e185eb7")
sender: "vivek"
receiver: "arunpravin"
message: "i am good pravin"
timestamp: 2021-08-05T15:44:26.348+00:00
```

>

```
_id: ObjectId("610c076070b3a2e41e185eb8")
sender: "vivek"
receiver: "arunpravin"
message: "
how are you doing?"
timestamp: 2021-08-05T15:44:32.450+00:00
```

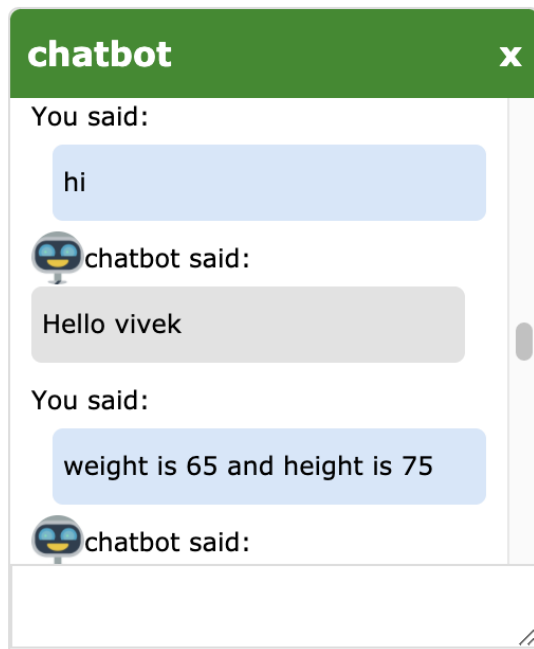


< PREVIOUS

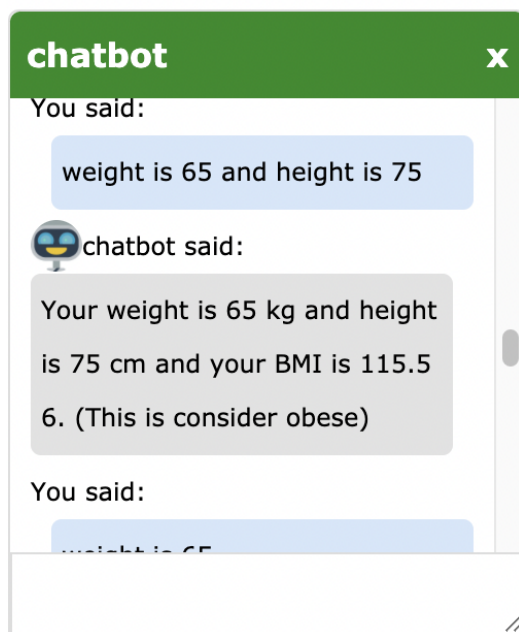
41-42 of 42 results

NEXT >

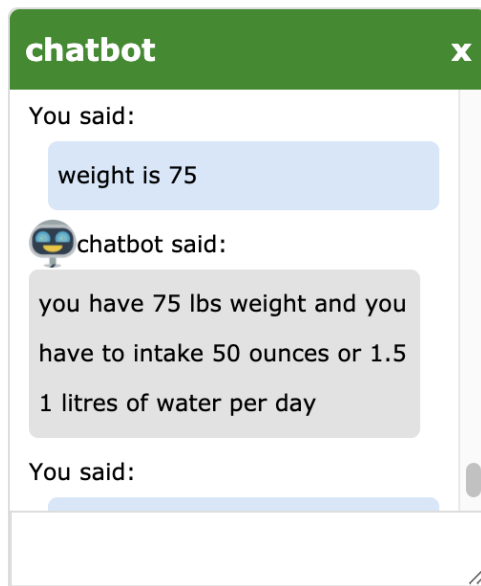
## Chatbot:



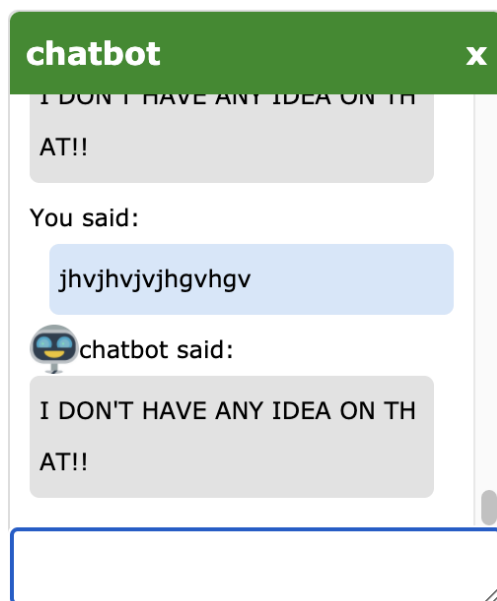
## Microservice 1:



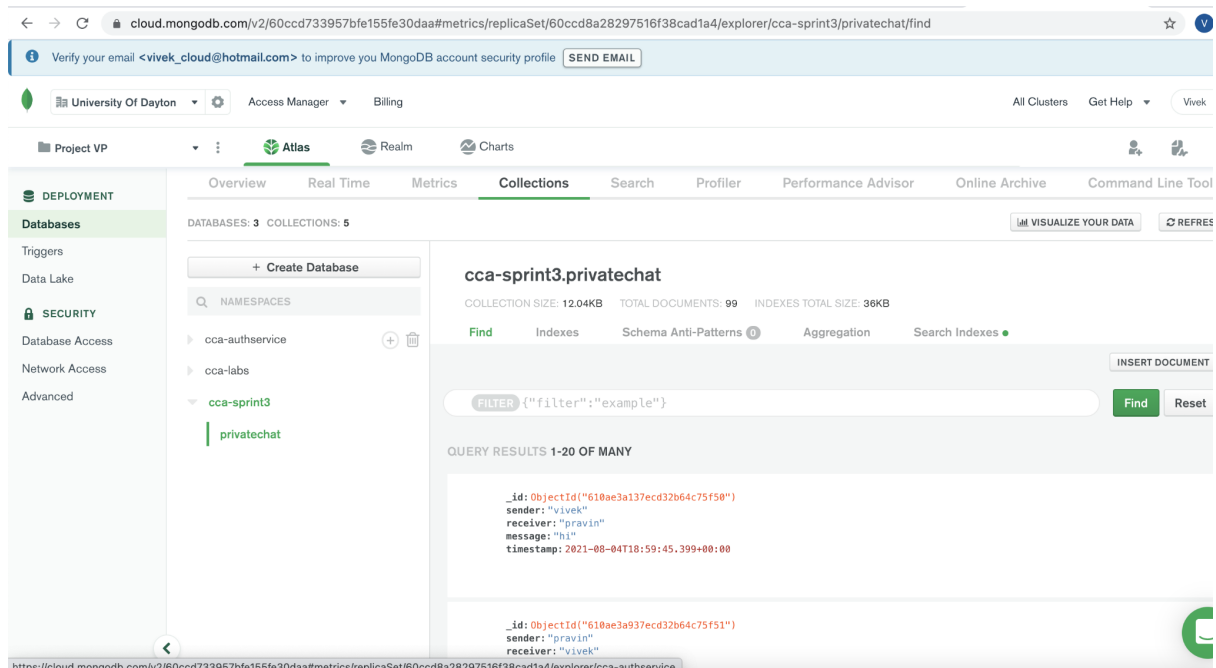
## Microservice 2:



## General question..



## New database in MongoDB for chat store and retrieve



## Appendix A:

### Chatclient.html

```
<!DOCTYPE html>
<html lang="en">

<head>

  <meta charset="utf-8">
  <link rel=stylesheet href="https://udayton-cloud.bitbucket.io/style1.css">
  <title>CCA-Test page of Vivek Manikandan</title>
  <script src="https://udayton-cloud.bitbucket.io/clock.js"></script>
  <script src="https://code.jquery.com/jquery-3.5.1.min.js"
crossorigin="anonymous"></script>
  <script src="/socket.io/socket.io.js"></script>
  <script>
    var socketio = io();
  </script>
  <style>
```

```

.owner{
    background-color: #4CAF50;
    font-family:calibri;
    color: white;
    padding: 5px;
    text-align: center;
    text-decoration: none;
    font-size: 12px;
    margin: 4px 2px;
    cursor: pointer;
}

#typinganimation {
    display : none
}

.tiblock {
align-items: center;
display: flex;
height: 17px;
}

.ticontainer .tidot {
    background-color: #90949c;
}

.tidot {
    -webkit-animation: mercuryTypingAnimation 1.5s infinite ease-in-out;
    border-radius: 2px;
    display: inline-block;
    height: 4px;
    margin-right: 2px;
    width: 4px;
}

@-webkit-keyframes mercuryTypingAnimation{
0%{
    -webkit-transform:translateY(0px)
}
28%{
    -webkit-transform:translateY(-5px)
}
44%{
    -webkit-transform:translateY(0px)
}
}

```

```

.tidot:nth-child(1){
-webkit-animation-delay:200ms;
}
.tidot:nth-child(2){
-webkit-animation-delay:300ms;
}
.tidot:nth-child(3){
-webkit-animation-delay:400ms;
}
.signin-page{
    width: 360px;
    padding: 8% 0 0;
    margin: auto;
}
.waiting{
    display: none;
}
.form{
    position: relative;
    z-index: 1;
    background: #FFFFFF;
    max-width: 360px;
    margin: 0 auto 100px;
    padding: 45px;
    text-align: center;
}
.form .signup-form{
    display: none;
}
.form .login-form{
    display: block;
}
.formchat .chat-form{
    display: none;
}

img {
    border-radius: 50%;
    width:40px;
}
.form input{
    font-family: calibri;
    outline: 0;
    background: #f2f2f2;

```



```
    width: 100%;
    border: 0;
    margin: 0 0 15px;
    padding: 15px;
    box-sizing: border-box;
    font-size: 15px;
}

.form .button{
    font-family: calibri;
    text-transform: uppercase;
    outline: 0;
    background: #4CAF50;
    width: 88%;
    border: 0;
    padding: 15px;
    color: #FFFFFF;
    font-size: 14px;

}

.form .message{
    margin: 15px;
    color: #4CAF50;
    text-decoration: none;
}

.button{
    background-color: #c94c4c;
    border: none;
    color: white;
    font-family:calibri;
    padding: 5px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 12px;
    margin: 4px 2px;
    cursor: pointer;
}

h2{color:white;
text-align:center;
font-family:calibri;}

h3{text-align:center;
font-family:calibri;}

p{text-align:center;
```

```

font-family:calibri;
font-size: 18px;}
.round{border-radius: 8px;}
#response{background-color: #ff9800;}

</style>
</head>
<body>

<div class="container wrapper">
  <div id="top">
    <div class= "owner">
      <h2><b><u>CPS 592 CLOUD COMPUTING AND APPLICATION</u><b></h2>
      <h1><b>A test webpage for cloud computing and applications - Project
Sprint 3</b></h1>
      <h2><b>By Vivek Manikandan & Arun Pravin Arunachalam Sakthivel</b></h2>
    </div>
    <!--div id = "email" onclick="showhideemail()">Show my email</div-->
    <div id = "user"></div>
    <script src = "email.js"></script>
    <div id="welcome"></div>
  </div>
  <div class="wrapper">
    <div id="menubar">
      <p>Private Web Chat</p>
      <canvas id = "analog-clock" width = "150" height = "150"
style="background-color:#999"></canvas>
      <div id="chat-sidebar"></div>
    </div>
    <div class = "formchat">
      <form class="chat-form">
        Chat message: <input name= "data" onkeypress =
"checkEnter(event)" onkeyup="socketio.emit('typing')" id="data">
        <input class="button round" type= "button" value="Send"
onclick="send()" ">
        <div id = "typing"></div>
        <div id="typinganimation">
          <div class="ticontainer">
            <div class="tiblock">typing.
              <div class="tidot"></div>
              <div class="tidot"></div>
              <div class="tidot"></div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>

```

```

        <div id = "online"></div>
        <div id="response"></div>
    </form>

</div>
</div>

<div id="main">
    <div class = "signin-page">
        <div class = "form">
            <form class = "Login-form">
                <input type = "text" placeholder = "Your Username" required
id = "signin_username"/>
                <input type = "password" placeholder = "password" required
id = "signin_password"/>
                <div class = "button" onclick="signin()">Login</div>
                <div class="waiting">
                    <p class="message">Please wait ...</p>
                    <div class="loader"></div>
                </div>
                <p class = "message">Not registered? <a href = "#" onclick =
"$($('.Login-form').hide();$('.signup-form').show())">create an account</a></p>
            </form>
            <form class="signup-form">
                <input type = "text" placeholder = "fullname" id =
"fullname1">
                <input type = "text" placeholder = "username" id =
"username1">
                <input type = "text" placeholder = "email address" id =
"email1">
                <input type = "password" placeholder = "password" id =
"password1">
                <div class = "button" onclick = "signup()" >create</div>
                <p class = "message">Already signedup? <a href='#' onclick =
"$($('.Login-form').show();$('.signup-form').hide())">Sign in</a> </p>
            </form>
        </div>
    </div>
</div>

<!--form action= "echo.php" method= "POST">

```

```

    Chat message: <input name= "data" onkeypress = "checkEnter(event)"
onkeyup="socketio.emit('typing')" id="data">
    <input class="button round" type= "button" value="Send" onclick="send()">
    <input class="button round" type= "button" value="jQuery Ajax GET"
onclick="jQueryAjax()"-->
    <!--input class="button round" type= "button" value="jQuery Ajax POST"
onclick="jQueryAjaxPost()"-->
<!--/form-->
<script>
    var canvas = document.getElementById("analog-clock");
    var ctx = canvas.getContext("2d");
    var radius = canvas.height / 2;
    ctx.translate(radius, radius);
    radius = radius * 0.90;
    setInterval(drawClock, 1000);

    function drawClock(){
        drawFace(ctx, radius);
        drawNumbers(ctx, radius);
        drawTime(ctx, radius);
    }
</script>
<script>
    function send(){
        var input = $("#data").val();
        if (input.length == 0) return;
        socketio.emit("message",input);
        $("#data").val("");
    }

    socketio.on("message" , (data) => {
        $("#response").append(data + "<br>");
    });
    socketio.on("typing", function(data) {
        //$("#typing").html(data);
        $("#typinganimation").show();
        setTimeout(()=>{$("#typinganimation").hide()},10*1000);
    });

    socketio.on("online", (data) => {
        $("#online").html(data + "<br>");
    });

</script>
<script>

```

```

function checkEnter(event){
    if (event.keyCode == 13) send();
}
</script>
<script>
    function signup(){
        var username = $("#username1").val();
        var password = $("#password1").val();
        var fullname = $("#fullname1").val();
        var email = $("#email1").val();

        if (username.length < 4 ){
            alert ("invalid inputs"+username);
            return
        }
        if (password.length < 4 ){
            alert ("invalid password"+password)
        }
        //var account = {username1 : username1, password1 : password1,
        fullname1:fullname1, email1:email1};
        socketio.emit("create", fullname,email,username,password);
    }
    socketio.on("login", (data) => {
        alert("the user signed in successfully"+data.username);
        $('head').append('<link href="chatbox.css" rel="stylesheet">');
        $('head').append('<script src="chatbox.js"></script>');
        const avatar = data.avatar ||
'https://i.pravatar.cc/50?u='+data.username;
        $('#welcome').html('Welcome ' +
data.fullname)
        $('#.waiting').hide();

        $('#.Login-form').hide();
        $('#.chat-form').show();
        setTimeout(()=>{
            setCurrentUser(data)
        },
        1000);

    });
    socketio.on("signup", (data)=>{
        alert("sign up successful"+data);
        $('#.signup-form').hide();
        $('#.Login-form').show();
    });

```

```

});
socketio.on("signup error", (data)=>{
    alert("Error while signing up!!!! tryagain!!!" + data);
});
</script>
<script>
    function signin(){
        var s_username = $("#signin_username").val();
        var s_password = $("#signin_password").val();

        if(!s_username || !s_password){
            alert("Type the credentials without leaving blank");
        }
        else{
            //var signin_cred = {username2:s_username,password2:s_password};
            socketio.emit("loginuser", s_username, s_password);
            $('#.waiting').show();
            console.log("end with the signin for web!!!!!!")
        }

    }
    socketio.on("login-error", (data)=>{
        alert("Login failed - Username not found!!" + data);
        $('#.waiting').hide();
    })
</script>

<script>
    function sendPrivateChat(receiver, message){
        socketio.emit("privatechat", receiver, message, currentUser);
    }

    function requestChatHistory(receiver){
        console.log("the chat history is username" + receiver +
"currentuser " + currentUser)
        socketio.emit("chathistory", receiver, currentUser);
    }

    socketio.on("onlineusers", function(userlist){
        setTimeout(() =>{
            displayUserList(userlist);
        },
        2000);
    });

```

```

});
socketio.on("updateusers",function(userlist){
    setTimeout(()=>{
        updateUserList(userlist)
    },
    100);
});
socketio.on("privatechat", (sender,message) => {
    displayFriendMessage(sender,message);
});

socketio.on("chathistory", (sender,message) => {
    console.log(sender, message)
    if (message.length == 0){
        console.log("No data found");
    }
    for (let x in message){
        chat = message[x];
        //console.log("the chat data of on chat history is
"+chat.sender+"receiver"+" "+chat.receiver+" "+message+" "+chat.message+
"current user" + " "+currentUser )
        if(chat.sender == sender){
            displayFriendMessage(chat.sender,chat.message);
        }
        else{
            displaySelfMessage(chat.receiver,chat.message);
        }
    }
});
</script>

</body>
</html>

```

## Index.js

```

const express = require('express')
let chatdb = require("./chatdb");
let chatbot = require("./chatbot");
const app = express()
const axios = require('axios');
var port = process.env.PORT || 8081;
app.use(express.static('static'))
const cors = require('cors')

```

```

app.use(cors())
const server = require('http').createServer(app);
const io = require('socket.io')(server);
//app.use(express.urlencoded({extended: false}))
server.listen(port, () =>
  console.log('HTTP server with express js listening to port'+port )
)
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/static/chatclient.html');
})

function userlist(event) {
  var sockets = io.sockets.sockets;
  let onlineUsers = new Map();
  let account = {
    username : "chatbot",
    fullname : "chatbot",
    email : "chatbot@us.com",
    avatar :
'https://image.flaticon.com/icons/png/512/740/740110.png'
  };
  onlineUsers.set("chatbot", account)
  for (var id in sockets) {
    const socketclient = sockets[id];
    if(socketclient && socketclient.authenticated &&
socketclient.profile) {
      onlineUsers.set(socketclient.username, socketclient.profile);
    }
  }

  let onlineUsersJSON = [];
  onlineUsers.forEach((value, key) => {
    onlineUsersJSON.push(value);
  });
  if(event == "login") {
    for (var id in sockets) {
      const socketclient = sockets[id];
      if(socketclient && socketclient.authenticated &&
socketclient.profile) {
        socketclient.emit("onlineusers", onlineUsersJSON)
      }
    }
  }
  if(event == "disconnect") {
    for (var id in sockets) {

```



```

        const socketclient = sockets[id];
        if(socketclient && socketclient.authenticated &&
socketclient.profile){
            socketclient.emit("updateusers", onlineUsersJSON)
        }
    }
}

}

io.on('connection', (socketclient) => {
    console.log('A new client is connected!');
    var onlineClients = Object.keys(io.sockets.sockets).length;
    console.log("the online clients count is
"+Object.keys(io.sockets.sockets).length);

    socketclient.on("message", (data) => {
        if(socketclient.authenticated == true)
        {
            console.log('Data from client: '+data);
            BroadcastAuthenticatedClients("message", `${socketclient.username} says:
${data}`);

        }

    });

    socketclient.on("loginuser", (username, password) => {

        let postData = {
            username: username,
            password: password
        }

        console.log(postData);
        axios.post("https://cca-sprintauth.herokuapp.com/login", postData)
            .then(function (response) {
                console.log("success");
                socketclient.authenticated = true;
                socketclient.username = postData.username;
                let user = response.data["account"];
                socketclient.profile = user;
                console.log("the user data is"+user.fullname);
                //userlist(user)
            })
    })
})

```

```

        socketclient.emit("login",user);
        userlist("login")
        var welcomemessage = `${socketclient.username} is connected! :
Number of connected clients : ${onlineClients}`
        console.log(welcomemessage);
        io.emit("online", welcomemessage);

    })

    .catch(function (response) {
        console.log(response.data);
        socketclient.emit("login-error", username);

    });

});

socketclient.on("create", (fullname, email, username, password) => {
    console.log("Got inside the create");
    let postData = {
        username: username,
        password: password,
        fullname: fullname,
        email: email
    }

    console.log(postData);
    axios.post("https://cca-sprintauth.herokuapp.com/signup", postData)
        .then(function (response) {
            console.log("Success");
            socketclient.authenticated = true;
            socketclient.username = postData.username;
            io.account = postData;
            socketclient.emit("signup");
        })
        .catch(function (response) {
            console.log(response.data);
            io.emit("signup-error", "User already exist please login");
        });

});

socketclient.on("typing", () => {
    if(socketclient.authenticated == true)
    {

```

```

        console.log("someone is typing");
        io.emit("typing", `${socketclient.username} is typing.....`);
    }

});

socketclient.on("privatechat", (receiver,message,sender) => {
    var sockets = io.sockets.sockets;
    for (var id in sockets){
        let socketclient = sockets[id];
        if (receiver == "chatbot"){
            console.log("chatbot on !!!!!!!")
            let privatechat = {sender:sender,receiver:receiver,message:message}
            chatdb.storePrivateMessage(privatechat)
            if (message.includes("weight") && message.includes("height")){
                socketclient.emit("privatechat",receiver,"LOADING!!!");
            }
            if (message.includes("weight") && !message.includes("height")){
                socketclient.emit("privatechat",receiver,"LOADING!!!");
            }
        }
        chatbot.chatbot_auto(sender,message,(data)=> {
            if(data){
                socketclient.emit("privatechat",receiver,data);
                privatechat = {sender:receiver,receiver:sender,message:data}
                chatdb.storePrivateMessage(privatechat)
            }
        });
    }

    if(socketclient && socketclient.authenticated && socketclient.username ==
receiver){
        let privatechat = {sender:sender,receiver:receiver,message:message}
        chatdb.storePrivateMessage(privatechat)
        socketclient.emit("privatechat",sender,message);
    }
}

});

socketclient.on("chathistory", (receiver,sender) => {
    //console.log("inside chat history",sender);
    //console.log("inside chat history",receiver)
    var sockets = io.sockets.sockets;
    for (var id in sockets){
        let socketclient = sockets[id];

```

```

        if(socketclient && socketclient.authenticated && socketclient.username ==
sender) {
            console.log("load private message");
            chatdb.loadPrivateMessage(sender,receiver,(message)=> {
                if(message) {
                    //console.log("the callback is from
index"+JSON.stringify(message));
                    socketclient.emit("chathistory",receiver,message);
                }
            });
        }
    }
});

socketclient.on('disconnect', () => {
    var onlineClients = Object.keys(io.sockets.sockets).length;
    var byemessage = `${socketclient.username} is disconnected: Number of
connected clients ${onlineClients}`
    console.log(byemessage);
    io.emit("online",byemessage);
    userlist("disconnect")
})

});

function BroadcastAuthenticatedClients(event,message) {
    var sockets = io.sockets.sockets;
    for (var id in sockets) {
        const socketclient = sockets[id];
        if(socketclient&&socketclient.authenticated) {
            socketclient.emit(event,message)
        }
    }
}

```

## Chatdb.js

```

const MongoClient = require('mongodb').MongoClient;

```

```

const mongourl =
"mongodb+srv://CCA-VIVEK-CLOUD:vivek8080@cca-vivek-cloud.djoox.mongodb.net/cca-spri
nt3?retryWrites=true&w=majority"
const dbClient = new MongoClient(mongourl, {useNewUrlParser: true,
useUnifiedTopology: true});

console.log("chat db on inside JS!!!!!!!!!!")

dbClient.connect(err => {
  if (err) throw err;
  console.log("Connected to the Mongo Db cluster");
})

module.exports.storePrivateMessage = (privatechat) => {
  const db = dbClient.db();
  console.log("inside private chat chatdb.js"+privatechat.sender);
  console.log("inside private chat chatdb.js"+privatechat.receiver);
  console.log("inside private chat chatdb.js"+privatechat.message);
  if (!privatechat.sender || !privatechat.receiver){
    console.log('SENDER/RECEIVER INVALID chat db');
  }
  else{
    let d = new Date();
    console.log("the date is"+d);

    let chatdata =
{sender:privatechat.sender,receiver:privatechat.receiver,message:privatechat.messag
e,timestamp:d}

    db.collection("privatechat").insertOne(chatdata, (err) => {
      if (err){
        console.log("CHAT DB DATA INSERT FAILED");
      }
      console.log("CHAT DB DATA INSERT successful!!!!!!!!");
    })
  }
}

module.exports.loadPrivateMessage = (sender,receiver,callback) => {
  const db = dbClient.db();
  console.log("inside load private chat");
  console.log("the sender is "+sender);
  console.log("the receiver is"+receiver);
  if (!sender || !receiver){
    console.log("LOAD PRIVATE CHAT SENDER/RECEIEVER NOT FOUND")
  }
}

```

```

    }
    db.collection("privatechat").find({
    $or:[ {sender:sender, receiver:receiver},
    {sender:receiver, receiver:sender}]
    }).limit(100).toArray(function (err, message) {
        if(err) throw err;
        console.log("load chat message"+JSON.stringify(message))
        callback(message);
    })
}

```

## Chatbot.js

```

const axios = require('axios');
console.log("Inside chatbox.js")

module.exports.chatbot_auto = (sender,message,callback)=>{
    if (message == "hi"){
        callback("Hello"+" "+sender)
    }
    else if(message.includes ("BMI") || (message.includes("bmi"))) {
        callback("Sure!! please provide me your weight in kgs and height in cms");
    }
    else if(message.includes ("water") || (message.includes("intake"))) {
        callback("Sure!! please provide me your weight in kgs to suggest the correct water consumption")
    }
    else if (message.includes("weight") && message.includes("height")){
        console.log("Message contains height and weight")
        let weight_fetch = "weight is"
        let re = new RegExp(weight_fetch + "\\s*(\\d+)");
        let kg = message.match(re);
        console.log("the weight given in message is "+kg[1])
        let height_fetch = "height is"
        let reg = new RegExp(height_fetch + "\\s*(\\d+)");
        let cm = message.match(reg);
        console.log("the height given in message is "+cm[1])

        axios.get("https://cca-cloud-microservices.herokuapp.com/bmi_calc?height="+cm[1]+"&weight="+kg[1])
            .then(function (response) {
                console.log("Success for bmi microservice" +response.data);
                callback(response.data)
            })
    }
}

```

```

    })
    .catch(function (response) {
        console.log("error caught"+response.data);
    });
}
else if (message.includes("weight") && !message.includes("height")){
    console.log("Message contains only weight")
    let weight_fetch = "weight is"
    let re = new RegExp(weight_fetch + "\\s*(\\d+)");
    let kg = message.match(re);
    console.log("the weight given in message is "+kg[1])

    axios.get("https://cca-sprint1-waterintake.azurewebsites.net/water_intake?weight="+
kg[1])

        .then(function (response) {
            console.log("Success for bmi microservice waterintake"
+response.data);

            callback(response.data)
        })
        .catch(function (response) {
            console.log("error caught"+response.data);
        });

    }
    else if (message .includes ("how are you")){
        callback("i am doing good. Thanks for asking,"+ " "
+sender.toUpperCase()+"!!!")
    }
    else{
        callback("I DON'T HAVE ANY IDEA ON THAT!!")
    }
}

```