In depth how to..

REAL TIME PRIVATE CHATTING APP USING ANGULAR 2, NODEJS, MONGODB AND SOCKET.IO

Shashank Tiwari

Author at www.codershood.info

CONTENTS

PREFACE

CONVENTIONS USED IN THIS BOOK

In this book,

1. We will use ES6 JavaScript in Nodejs and for Angular 2, we will use Typescript.
2. Throughout this tutorial, Angular means Angular version greater than 2.x unless stated otherwise.
3. I am also assuming the reader of this e-book has intermediate understanding of JavaScript, Angular and solid understanding of HTML and CSS.

CODE SAMPLES

You can download the code sample from below URL.

Download the code : https://drive.google.com/open?id=0B99SY51zz9asb2poZzU5YWJIRFE

INTRODUCTION

WHY PRIVATE CHAT APPLICATION?

I have seen people writing about Chatting application, but most of them are conference chat or just a group chat where all the online users can chat with each other. I personally never found a good tutorial on private chatting application. So that gave me motivation to write this e-book.

WHAT WE ARE GOING TO BUILD?

In this application, we will create a private chatting application using Nodejs as a server side language with MongoDB as database in order to store messages. Here Angular will be used as client side language, where I will cover some its cool features. Down the road I will use some of the third party module in Angular.

For the real-time updates of chat list and occurrence of messages we will use socket.io.

1. CREATING A NODEJS SERVER

DIRECTORY STRUCTURE OF THE SERVER:

Below you can see the directory structure for our server. In the /utils folder I have 5 files as you can see below.

DIRECTORY STRUCTURE:

```
+--- node_modules
|
+--- utils
|
|     +-- config.js
|     +-- db.js
|     +-- helper.js
|     +-- routes.js
|     +-- scoket.js
+-- server.js
+-- package.json
```

PACKAGE.JSON:

Do I even need to talk about this file! Okay, for those who don't know much about this file; This file contains the information of our Nodejs project. For example, Name of Project, dependencies and developer dependencies required in our project, information related to testing of your Project as so on.

SERVER.JS:

This file entry point for our application. In this file, we will require all the useful files for our application. For example, here we will initiate a call to include the routes and the socket events of our application.

And at the end, we will create and start a Nodejs server.Now let's take a look into utils folder. Here we will have 5 files listed below, each file is used for its own purpose.

- config.js
- db.js
- helper.js
- routes.js
- scoket.js

CONFIG.JS:

In this file, I have written all the application configuration required for our application. For example, from where to fetch static files or what is the default view engine.

DB.JS:

This file is used for handling the database connections. Here I am connecting Nodejs application to MongoDB. To connect the MongoDB, we will use MongoDB driver which available for Nodejs, we will talk about this in the coming chapter.

HELPER.JS:

This file is the heart of our application, this file will perform all the operation for our application. From storing messages to sending list of online users, Login registration almost everything.

ROUTES.JS:

This file contains all the express routes of our chat applications. routes.js consumes helper.js and performs all the tasks.

SOCKET.JS:

Well, we are creating a chat application and its very obvious that it needs to be Real-time. So for our application, socket.js comes into the resume. This file contains all the socket event of our chat applications and send the real-time updates to the user.

CREATING SERVER.JS

Before creating a server let's take a look at our package.json file, in the previous section we learned about it.

PACKAGE.JSON:

```json
JSON

{
  "name": "rest-api",
  "version": "0.0.1",
  "description": "This Rest API for MY Angular2 Projects.",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "rest",
    "api"
  ],
  "author": "Shashank Tiwari",
  "license": "MIT",
  "dependencies": {
    "async": "^2.1.5",
    "body-parser": "^1.15.2",
    "cors": "^2.8.1",
    "express": "^4.14.0",
    "mongodb": "^2.2.19",
    "socket.io": "^1.7.2"
  }
}
```

In the above file, as you can see we have 6 dependencies for our application.

First step you have to do is, create a Nodejs project. The best way to create a new Nodejs project is start from below command.

```bash
BASH

npm init
```

And install the dependencies manually.

OR

You can just copy the above **package.json** file, run the below command, The below command will install the dependencies for you.

```bash
BASH

npm install
```

Alright, So by now I assume you have created package.json file for your application. Now let’s create a server.js file, which will be the entry point for our application.

So create **server.js** file inside the root directory of your project and write down the below code. In the server.js file, I will include nodejs modules as well as file shown in utils folder.

SERVER.JS:

```javascript
JAVASCRIPT
'use strict';

const express = require("express");
const http = require('http');
const socketio = require('socket.io');
const bodyParser = require('body-parser');
const cors = require('cors');

const socketEvents = require('./utils/socket');
const routes = require('./utils/routes');
const config = require('./utils/config');


class Server{

  constructor(){
    this.port =  process.env.PORT || 4000;
    this.host = `localhost`;

    this.app = express();
    this.http = http.Server(this.app);
    this.socket = socketio(this.http);
    }

    appConfig(){
        this.app.use(
            bodyParser.json()
        );
        this.app.use(
            cors()
        );
        new config(this.app);
    }

    /* Including app Routes starts*/
    includeRoutes(){
        new routes(this.app).routesConfig();
        new socketEvents(this.socket).socketConfig();
    }
    /* Including app Routes ends*/

    appExecute(){

        this.appConfig();
        this.includeRoutes();

        this.http.listen(this.port, this.host, () => {
            console.log(`Listening on http://${this.host}:${this.port}`);
        });
    }

}

const app = new Server();
app.appExecute();
```

2.CONNECTING NODEJS TO MONGODB

In this chapter we will connect our Nodejs application to MongoDB. I won't talk much about MongoDB here, but yes I will show you How we can use Nodejs MongoDB driver to connect MongoDB database.

Create a db.js file inside **/utils** folder and write down the below code. In the below code I am using mongodb driver connect MongDB to Nodejs.

DB.JS:

```javascript
JAVASCRIPT
"use strict";
/* requiring MongoDB module */
const mongodb = require('mongodb');
const assert = require('assert');

class Db{

  constructor(){
    this.mongoClient = mongodb.MongoClient;
    this.ObjectID = mongodb.ObjectID;
    this.mongoURL = `mongodb://127.0.0.1:27017/local`;
  }

  onConnect(callback){
    this.mongoClient.connect(this.mongoURL, (err, db) => {
      assert.equal(null, err);
      callback(db,this.ObjectID);
    });
  }
}
module.exports = new Db();
```

CHAPTER 3. CREATING REST API ENDPOINTS

In this chapter, we will create the endpoints for our application. First we will list down the endpoints and the usage of them and after that we will use express routing to create endpoint.

In this section we will get to know, how many endpoints we will require and so to implement them create a **route.js** in /utils folder.

LISI OF API ENDPOINTS

Below is the list of endpoints that we are going to create along full explanation.

/usernameCheck: This endpoint is to check the availability of username while registration of the user. In our application I am keeping the username of each and every user unique.

So every time when user type something in the username text box, an Ajax will execute and check if the given username is available or not.

PARAMETERS: **username**

RESPONSE:

```json
{
   error : false,
   message: 'This username is available.'
}
```

/registerUser: To register the new user into the system. As of now I am no storing the user's profile image. But may in next version of this e-book, I will add that feature too.

PARAMETERS: **username** , **email** , **password**

RESPONSE:

```
                                                                    JSON
{
   error : false,
   userId : ‘0987545hsf45j990l’,
   message: ‘User registration successful.’
}
```

/login: As the name suggests we will use this endpoint for login.

PARAMETERS: **username** , **password**

RESPONSE:

```
                                                                    JSON
{
   error : false,
   message: ‘User logged in.’
}
```

/userSessionCheck: Using this endpoint we can check, if user is logged in or not!

PARAMETERS: **userId**

RESPONSE:

```
                                                                    JSON
{
   error : false,
   username:’shashank’
   message: ‘User logged in.’
}
```

/getMessages: This end point will give the conversation between the two users.

PARAMETERS: **userId** , **toUserId**

RESPONSE:

```
                                                          JSON
{
   error : false,
   messages: [
   ]
}
```

/getUsersToChat: This end point will give us list of users with whom we can start a new chat.

PARAMETERS: **userId**

RESPONSE:

```
                                                          JSON
{
   error : false,
   messages:[
   ]
}
```

CREATING ROUTES.JS FILE

Now I showed the list of endpoints and the usage of them. Now it’s time to create those endpoints.

So to do that create **routes.js** file inside the **/utils** folders.

ROUTES.JS:

```javascript
'use strict';
const helper = require('./helper');
class Routes{
  constructor(app){
    this.app = app;
  }
  /*
  * Creating app Routes starts
  */
  appRoutes(){
    /* Creating route to check the availability of username starts*/
    this.app.post('/usernameCheck',(request,response) =>{
      if (request.body.username === "") {
        response.status(412).json({
          error : true,
          message : `username cant be empty.`
        });
      } else {
        helper.userNameCheck( {
          username : request.body.username.toLowerCase()
        }, (count)=>{
          let result = {};
          if (count > 0) {
            result.error = true;
            result.message = 'This username is alreday taken.';
            response.status(401).json(result);
          } else {
            result.error = false;
            result.message = 'This username is available.';
```

```
                response.status(200).json(result);
            }
        });
    }
});
/* Creating route to check the availability of username ends*/
/* Creating route, to register the user starts*/
this.app.post('/registerUser',(request,response) =>{
    const data = {
        username : (request.body.username).toLowerCase(),
        email : request.body.email,
        password : request.body.password
    };
    let registrationResponse = {}
    if(data.username === '') {
        registrationResponse.error = true;
        registrationResponse.message = `username cant be empty.`;
        response.status(412).json(registrationResponse);
    }else if(data.email === ''){
        registrationResponse.error = true;
        registrationResponse.message = `email cant be empty.`;
        response.status(412).json(registrationResponse);
    }else if(data.password === ''){
        registrationResponse.error = true;
        registrationResponse.message = `password cant be empty.`;
        response.status(412).json(registrationResponse);
    }else{
        data.timestamp = Math.floor(new Date() / 1000);
        data.online = 'Y' ;
        data.socketId = '' ;
        helper.registerUser( data, (error,result)=>{
            if (error) {
                registrationResponse.error = true;
                registrationResponse.message = `User registration unsuccessful,try after some time.`;
                response.status(417).json(registrationResponse);
            }else{
                registrationResponse.error = false;
                registrationResponse.userId = result.insertedId;
                registrationResponse.message = `User registration successful.`;
                response.status(200).json(registrationResponse);
            }
        });
    }
});
/* Creating route, to register the user ends*/
/* Creating route, to Login the user starts*/
this.app.post('/login',(request,response) =>{
    const data = {
        username : (request.body.username).toLowerCase(),
        password : request.body.password
    };
    let loginResponse = {}
    if(data.username === '' || data.username === null) {
        loginResponse.error = true;
        loginResponse.message = `username cant be empty.`;
        response.status(412).json(loginResponse);
    }else if(data.password === '' || data.password === null){
        loginResponse.error = true;
        loginResponse.message = `password cant be empty.`;
        response.status(412).json(loginResponse);
    }else{
        helper.login( data, (error,result)=>{
            if (error || result === null) {
                loginResponse.error = true;
                loginResponse.message = `Invalid username and password combination.`;
                response.status(401).json(loginResponse);
            }else{
                loginResponse.error = false;
                loginResponse.userId = result._id;
                loginResponse.message = `User logged in.`;
                response.status(200).json(loginResponse);
            }
        });
    }
});
/* Creating route, to Login the user ends*/


/* Creating route, if user is logged in or not starts*/
```

```
this.app.post('/userSessionCheck',(request,response) =>{
  let userId = request.body.userId;
  let sessionCheckResponse = {}
  if (userId == '') {
    sessionCheckResponse.error = true;
    sessionCheckResponse.message = `User Id cant be empty.`;
    response.status(412).json(sessionCheckResponse);
  }else{
    helper.userSessionCheck( {
      userId : userId,
    }, (error,result)=>{
      if (error || result === null) {
        sessionCheckResponse.error = true;
        sessionCheckResponse.message = `User is not logged in.`;
        response.status(401).json(sessionCheckResponse);
      }else{
        sessionCheckResponse.error = false;
        sessionCheckResponse.username = result.username;
        sessionCheckResponse.message = `User logged in.`;
        response.status(200).json(sessionCheckResponse);
      }
    });
  }
});
/* Creating route, if user is logged in or not ends*/
/* Creating route, get the conversation between the two users starts*/
this.app.post('/getMessages',(request,response) =>{
  let userId = request.body.userId;
  let toUserId = request.body.toUserId;
  let messages = {}
  if (userId == '') {
    messages.error = true;
    messages.message = `userId cant be empty.`;
    response.status(200).json(messages);
  }else{

    helper.getMessages( userId, toUserId, (error,result)=>{
      if (error) {
        messages.error = true;
```

```
                messages.message = `Internal Server error.`;
                response.status(500).json(messages);
            }else{
                messages.error = false;
                messages.messages = result;
                response.status(200).json(messages);
            }
        });
    }
});
/* Creating route, get the conversation between the two users ends*/
/* Creating route, will give us list of users with whom we can start a new chat starts*/
this.app.post('/getUsersToChat',(request,response) =>{
    let userId = request.body.userId;
    let chatList = {}
    if (userId == '') {
        chatList.error = true;
        chatList.message = `userId cant be empty.`;
        response.status(200).json(chatList);
    }else{
        helper.getUsersToChat( userId, (error,result)=>{
            if (error) {
                chatList.error = true;
                chatList.message = `Internal server error.`;
                response.status(500).json(chatList);
            }else{
                chatList.error = false;
                chatList.chatList = result;
                response.status(200).json(chatList);
            }
        });
    }
});
/* Creating route, will give us list of users with whom we can start a new chat ends*/


this.app.get('*',(request,response) =>{
    response.sendFile(path.join(__dirname,'../dist/index.html'));
});
}
```

```
    routesConfig(){
        this.appRoutes();
    }
}
module.exports = Routes;
```

EXPLANATION: In the Route class, we have 3 methods constructor(), appRoutes() and routesConfig() respectively. The constructor method expects instance of express app. In the appRoutes() method I have implemented the all the endpoints using express. And in routesConfig(), am calling appRoutes() method which loads all the endpoints.

If you notice, I have already required the helper.js file and using the method of Helper class. Don't worry we will look into it in next to next chapter. And I assume that rest of the things are self-explanatory.

CHAPTER 4. INTRODUCING SOCKET.IO

In this chapter we will use socket.io module and implement the socket.io events. These events will send the response to the client from server.

LIST OF TOTAL SOCKET EVENTS

Let's first see the list of socket events and after that we will implement them.

CHAT-LIST: In this socket event I am sending the chat list of users, to the logged in user. The list will contain the user's id, name and some other useful stuff.

Here I am sending only send list of whose users with whom logged in user had a conversation. You can see the below response as an example.

```json
JSON

{
  "error": false,
  "singleUser": false,
  "chatList": [
   {
    "_id": "589ca01a12dc98198410ab80",
    "username": "Rajeev",
    "timestamp": 1486659610,
    "online": "N",
    "socketId": "z-_Ye-ZztTZqdUuxAAAE"
   },
   {
    "_id": "58b7011242d9aa20ecb08ac1",
    "username": "AJ",
    "timestamp": 1488388370,
    "online": "Y",
    "socketId": "EVBADWZiXayo3xKzAAAB"
   }
  ]
}
```

Here, the singleUser node indicates the list contains the information about single user or list contains information about multiple users and off course the value the datatype of singleUser node is Boolean.

ADD-MESSAGE: As the name suggests, in this socket event I am sending message when user sends to another user along with each other's user id. Below is the response for the same.

```json
JSON
{
 fromUserId: "58b7011242d9aa20ecb08ac1",
 message: "Hello Lady halena!",
 timestamp: 1494187501,
 toUserId: "589ca00e12dc98198410ab7f",
 username: "Rajeev",
 _id: "590f7dedd4cc8217d02ea8c2"
}
```

LOGOUT: In this event I am making a logout operation for user.

DISCONNECT: Here, in this event I am broadcasting the event, where all the logged in user will be notified that a specific user us logged out from the system. Below is the response for the same.

```json
JSON
{
 error: false,
 userDisconnected: true,
 socketId: "EVBADWZiXayo3xKzAAAB"
}
```

CREATING SOCKET.JS FILE

Now, we will create socket.js file again inside /utils folder. In socket.js file we will implement above discussed events and these events will push real-time data to the client side user.

SOCKET.JS:

```
JSON
'use strict';

const path = require('path');
const helper = require('./helper');

class Socket{

   constructor(socket){
      this.io = socket;
   }

   socketEvents(){

      this.io.on('connection', (socket) => {

         /**
         * get the user's Chat list
         */
         socket.on('chat-list', (data) => {

           let chatListResponse = {};

            if (data.userId == '') {

               chatListResponse.error = true;
               chatListResponse.message = `User does not exits.`;

               this.io.emit('chat-list-response',chatListResponse);

            }else{

               helper.getUserInfo( data.userId,(err, UserInfoResponse)=>{

                  delete UserInfoResponse.password;

                  helper.getChatList( data.userId ,socket.id, (err, response)=>{

                     this.io.to(socket.id).emit('chat-list-response',{
                        error : false ,
                        singleUser : false ,
                        chatList : response === null ? null : response.chatList
                     });

                     if (response !== null) {
                        let chatListIds = response.chatListIds;
                        chatListIds.forEach( (Ids)=>{
                           this.io.to(Ids).emit('chat-list-response',{
                              error : false ,
                              singleUser : true ,
                              chatList : [UserInfoResponse]
                           });
                        });
```

```
                }
            });
        });
    }
});
/**
* send the messages to the user
*/
socket.on('add-message', (data) => {

    if (data.message === '') {

        this.io.to(socket.id).emit(`add-message-response`,`Message cant be empty`);

    }else if(data.fromUserId === ''){

        this.io.to(socket.id).emit(`add-message-response`,`Unexpected error, Login again.`);

    }else if(data.toUserId === ''){

        this.io.to(socket.id).emit(`add-message-response`,`Select a user to chat.`);

    }else{

        let toSocketId = data.toSocketId;
        let fromSocketId = data.fromSocketId;
        delete data.toSocketId;
        data.timestamp = Math.floor(new Date() / 1000);

        helper.insertMessages(data,( error , response)=>{
            helper.getUserInfo(data.fromUserId,(error,userInfoResponse)=>{
                data.username = userInfoResponse.username;
                this.io.to(toSocketId).emit(`add-message-response`,data);
            });
        });
    }
});


/**
* Logout the user
*/
socket.on('logout',(data)=>{

    const userId = data.userId;

    helper.logout(userId , false, (error, result)=>{
        this.io.to(socket.id).emit('logout-response',{
            error : false
        });
        socket.disconnect();
    });
});


/**
* sending the disconnected user to all socket users.
*/
socket.on('disconnect',()=>{
    setTimeout(()=>{
        helper.idUserLoggedOut(socket.id,(response)=>{
            if (response.loggedOut) {
                socket.broadcast.emit('chat-list-response',{
                    error : false ,
                    userDisconnected : true ,
                socketId : socket.id
              });
            }
          });
        },1000);
      });

    });

  }

  socketConfig(){

    this.io.use(function(socket, next) {
      let userID = socket.request._query['userId'];
      let userSocketId = socket.id;
      const data = {
        id : userID,
        value : {
          $set :{
            socketId : userSocketId,
            online : 'Y'
          }
        }
      }

      helper.addSocketId( data ,(error,response)=>{
        next();
      });
    });

    this.socketEvents();
  }
}
module.exports = Socket;
```

Explanation:

In Socket class, I have created three methods contructor(), socketEvents() and socketConfig() respectively.

Let’s start with socketConfig() method. Whenever a new client connects to the socket server, the corresponding socketId is attached with the user, in other words socketid is inserted into mongoDB users collection.

Inside socketEvents() I have created all the necessary events to push the real-time data to the client.

CHAPTER 5. MAKING THINGS WORK

Till now, I have created all the required files. But, if you have noticed I have required helper.js file inside routes.js and socket.js. So in this chapter we will create a Helper() class.

The helper.js file contains all the important functions and logic behind all the operations that we are going to perform, in other words you can say this file is a heart of this application. The helper class file has lot of methods defined into it and we will look into each method one by one.

Now, inside helper class I have created more than 12 methods. However, I have written the comments, parameters and the response by each method prior to its initialization. And Here I am expecting from the reader of this e-book must have clear understanding of JavaScript and logic building.

CREATING HELPER.JS FILE

HELPER.JS:

```
JSON
'use strict';
const async = require('async');

class Helper{

  constructor(){
    this.Mongodb = require("./db");
  }

  /*
  * Name of the Method : userNameCheck
  * Description : To check if the username is available or not.
  * Parameter :
  *      1) data query object for MongDB
  *      2) callback function
  * Return : callback
  */
  userNameCheck(data,callback){
    this.Mongodb.onConnect( (db,ObjectID) => {
      db.collection('users').find(data).count( (err, result) => {
        db.close();
        callback(result);
      });
    });
  }

  /*
  * Name of the Method : login
  * Description : login the user.
  * Parameter :
  *      1) data query object for MongDB
  *      2) callback function
  * Return : callback
  */
```

```
login(data,callback){
   this.Mongodb.onConnect( (db,ObjectID) => {
      db.collection('users').findAndModify( data ,[], {$set: {'online': 'Y'}},{},(err, result) => {
         db.close();
         callback(err,result.value);
      });
   });
}

/*
* Name of the Method : registerUser
* Description : register the User
* Parameter :
*       1) data query object for MongDB
*       2) callback function
* Return : callback
*/
registerUser(data,callback){
   this.Mongodb.onConnect( (db,ObjectID) => {
      db.collection('users').insertOne(data, (err, result) =>{
         db.close();
         callback(err,result);
      });
   });
}

/*
* Name of the Method : userSessionCheck
* Description : to check if user is online or not.
* Parameter :
*       1) data query object for MongDB
*       2) callback function
* Return : callback
*/
userSessionCheck(data,callback){
   this.Mongodb.onConnect( (db,ObjectID) => {
      db.collection('users').findOne( { _id : ObjectID(data.userId) , online : 'Y'}, (err, result) => {
         db.close();
         callback(err,result);
      });
   });
}


/*
* Name of the Method : getUserInfo
* Description : to get information of single user.
* Parameter :
*       1) userId of the user
*       2) callback function
* Return : callback
*/
getUserInfo(userId,callback){
   this.Mongodb.onConnect( (db,ObjectID) => {
      db.collection('users').findOne( { _id : ObjectID(userId)}, (err, result) => {
         db.close();
         callback(err,result);
      });
   });
}

/*
* Name of the Method : addSocketId
* Description : Updates the socket id of single user.
* Parameter :
*       1) userId of the user
*       2) callback function
* Return : callback
*/
addSocketId(data,callback){
   this.Mongodb.onConnect( (db,ObjectID) => {
      db.collection('users').update( { _id : ObjectID(data.id)}, data.value ,(err, result) => {
         db.close();
         callback(err,result.result);
      });
   });
}
```

```
/*
* Name of the Method : getUsersToChat
* Description : To get the list of users to start a new chat.
* Parameter :
*       1) userId (socket id) of the user
*       2) callback function
* Return : callback
*/
getUsersToChat(userId,callback){

    const Mongodb = this.Mongodb;

    async.waterfall([
        (callback)=>{

            Mongodb.onConnect( (db,ObjectID) => {
                /*
                * Finding the list of userid from chatlist collection starts.
                */
                db.collection('chatlist').findOne( {'userId':userId} , (err, queryResponse) => {
                    db.close();
                    /*
                    * if loop starts
                    */
                    if (queryResponse === null ) {
                        callback(true,{
                            getAllUsers:true
                        });
                    }else{
                        callback(null,queryResponse);
                    }
                });
                /*
                * Finding the list of userid from chatlist collection ends.
                */
            });
        },
        (params, callback)=>{

            let chatListIds = params.chatlist;

            if (chatListIds.length === 0 || chatListIds.length < 1) {
                callback(true,{
                    getAllUsers:true
                });
            }else{

                Mongodb.onConnect( (db,ObjectID) => {
                    let mongoIds = [];
                    chatListIds.forEach( (ids)=>{
                        mongoIds.push(ObjectID(ids));
                    });
                    mongoIds.push(ObjectID(userId));

                    /*
                    * fetching the user's information from the users table.
                    * here, `mongoIds` is list of user ids which we fetched from chatlist ocllection.
                    */
                    db.collection('users').find({_id : { $nin : mongoIds } }).toArray( (err, queryResult) => {
                        db.close();

                        if (err) {
                            callback(true,{
                                getAllUsers:true
                            });
                        }else{
                            /*
                            * Removing the password and email from the Query result.
                            */
                            queryResult.forEach( (users)=>{
                                delete users.password;
                                delete users.email;
                            });

                            callback(true,{
                                getAllUsers:false,
                                chatlist : queryResult
                            });
                        }
                    });
                });
            }
        },
        ], (err, result)=>{
        if (result.getAllUsers) {
            Mongodb.onConnect( (db,ObjectID) => {
                db.collection('users').find({ _id : { $ne : ObjectID(userId) }}).toArray( (err, result) => {
                    db.close();
                    if (err) {
                        callback(true,null);
                    }else{
                        callback(false,result);
                    }
                });
            });
        }else{
            callback(false,result.chatlist);
        }
        });
        }

        /*
        * Name of the Method : getChatList
        * Description : To get the list of online user.
        * Parameter :
        *       1) userId (socket id) of the user
        *       2) callback function
        * Return : callback
        */
        getChatList(userId,userSocketId,callback){

        const Mongodb = this.Mongodb;

        async.waterfall([
        (callback)=>{

            Mongodb.onConnect( (db,ObjectID) => {
                /*
                * Finding the list of userid from chatlist collection starts.
                */
                db.collection('chatlist').findOne( {'userId':userId} , (err, queryResponse) => {
                    db.close();
                    /*
                    * if loop starts
                    */
                    if (queryResponse === null ) {
                        callback(true,{
                            isNull:true
                        });
                    }else{
                        callback(null,queryResponse);
                    }
                });
                /*
                * Finding the list of userid from chatlist collection ends.
                */
            });
        },
        (params, callback)=>{

            let chatListIds = params.chatlist;

            if (chatListIds.length === 0 || chatListIds.length < 1) {
                callback(true,{
                    isNull:true
                });
            }else{

                Mongodb.onConnect( (db,ObjectID) => {
                    let mongoIds = [];
                    chatListIds.forEach( (ids)=>{
                        mongoIds.push(ObjectID(ids));
                    });

                    /*
                    * fetching the user's information from the users table.
                    * here, `mongoIds` is list of user ids which we fetched from chatlist ocllection.
                    */
                    db.collection('users').find({_id : { $in : mongoIds } }).toArray( (err, queryResult) => {
                        db.close();

                        if (err) {
                            callback(true,{
                                isNull:true
                            });
                        }else{
                            /*
                            * Removing the password and email from the Query result.
                            */
                            let socketIds = [];
                            queryResult.forEach( (users)=>{
                                delete users.password;
                                delete users.email;
                                socketIds.push(users.socketId);
                            });

                            callback(true,{
                                isNull:false,
                                chatList : queryResult,
                                chatListIds : socketIds
                            });
                        }
                    });
                });
            }
        },
    ], (err, result)=>{
        if (result.isNull) {
            callback(true,null);
        }else{
            callback(false,result);
        }
    });
}

/*
* Name of the Method : insertMessages
* Description : To insert a new message into DB.
* Parameter :
*       1) data comprises of message,fromId,toId
*       2) callback function
* Return : callback
*/
insertMessages(data,callback){
    const Mongodb = this.Mongodb;
    const userId = data.fromUserId;
    const toUserId = data.toUserId;

    this.insertChatListForFriend(data);

    async.waterfall([
        (callback)=>{
            /*
            * Finding the userid from chatlist collection starts.
            */
            Mongodb.onConnect( (db,ObjectID) => {
                db.collection('chatlist').findOne( {'userId':userId} , (err, queryResponse) => {
                    db.close();
                    if (queryResponse === null ) {
                        /* Passing callback to next function */
                        callback(null,null);
                    }else{
                        /* Passing callback to next function */
                        callback(null,queryResponse);
                    }
                });
            });
            /*
            * Finding the userid from chatlist collection ends.
            */
        },
        (params,callback)=>{
            /*
            * If user is not Present in the chatlist collection,
            * then insert a New userId and Chatlit for that userId
            */
            if (params === null) {
                /* Inserting userId and list of ids starts*/
                Mongodb.onConnect( (db,ObjectID) => {
                    db.collection('chatlist').insertOne({
                        userId : userId,
                        chatlist : [ toUserId ]
                    }, (err, result) =>{
                        db.close();
                        callback(true,'queryResponse');
                    });
                });
                /* Inserting userId and list of ids starts*/
            }else{
                /*
                * If user is Present in the chatlist collection,
                * then update the Chatlit for that userId
                */
                let newChatIDs = params.chatlist;
                /* Updating userId and list of ids starts*/
                if(newChatIDs.indexOf(toUserId)){
                    newChatIDs.push(toUserId);
                    Mongodb.onConnect( (db,ObjectID) => {
                        db.collection('chatlist').findAndModify( {userId : userId} ,[], {$set: {chatlist: newChatIDs}},{},(err, result) => {
                            db.close();
                            callback(true,result.value);
                        });
                    });
                }else{
                    callback(true,null);
                }

                /* Updating userId and list of ids ends*/
            }
        }
    ],(params,result)=>{
        /* insert the message into the messages collection starts*/
        Mongodb.onConnect( (db,ObjectID) => {
            db.collection('messages').insertOne(data, (err, result) =>{
                db.close();
                callback(err,result);
            });
        });
        /* insert the message into the messages collection ends*/
    });
}

/*
* Name of the Method : insertChatListForFriend
* Description : To insert a chatlist for the friend.
* Parameter :
*       1) data comprises of message,fromId,toId
* Return : callback
*/
insertChatListForFriend(data){
    const Mongodb = this.Mongodb;
    const userId = data.fromUserId;
    const toUserId = data.toUserId;

    async.waterfall([
        (callback)=>{
            /*
            * Finding the userid from chatlist collection starts.
            */
            Mongodb.onConnect( (db,ObjectID) => {
                db.collection('chatlist').findOne( {'userId':toUserId} , (err, queryResponse) => {
                    db.close();
                    if (queryResponse === null ) {
                        /* Passing callback to next function */
                        callback(null,null);
                    }else{
                        /* Passing callback to next function */
                        callback(null,queryResponse);
                    }
                });
            });
            /*
            * Finding the userid from chatlist collection ends.
            */
        },
        (params,callback)=>{
            /*
            * If user is not Present in the chatlist collection,
            * then insert a New userId and Chatlit for that userId
            */
            if (params === null) {
                /* Inserting userId and list of ids starts*/
                Mongodb.onConnect( (db,ObjectID) => {
                    db.collection('chatlist').insertOne({
                        userId : toUserId,
                        chatlist : [ userId ]
                    }, (err, result) =>{
                        db.close();
                        callback(true,'queryResponse');
                    });
                });
                /* Inserting userId and list of ids starts*/
            }else{
                /*
                * If user is Present in the chatlist collection,
                * then update the Chatlit for that userId
                */
                /* Updating userId and list of ids starts*/
                let newChatIDs = params.chatlist;
                if(newChatIDs.indexOf(userId)){
                    newChatIDs.push(userId);
                    Mongodb.onConnect( (db,ObjectID) => {
                        db.collection('chatlist').findAndModify( {userId : toUserId} ,[], {$set: {chatlist: newChatIDs}},{},(err, result) => {
                            db.close();
                            callback(err,result.value);
                        });
                    });
                }
                /* Updating userId and list of ids ends*/
            }
        }
    ],(params,result)=>{
        //Write Any important code here.....
    });
}

/*
* Name of the Method : getMessages
* Description : To fetch messages from DB between two users.
* Parameter :
*       1) userId, toUserId
*       2) callback function
* Return : callback
*/
getMessages(userId, toUserId, callback){

    const data = {
        '$or' : [
            { '$and': [
                    {
                        'toUserId': userId
                    },{
                        'fromUserId': toUserId
                    }
                ]
            },{
                '$and': [
                    {
                        'toUserId': toUserId
                    }, {
                        'fromUserId': userId
                    }
                ]
            },
        ]
    };
    this.Mongodb.onConnect( (db,ObjectID) => {
        db.collection('messages').find(data).sort({'timestamp':1}).toArray( (err, result) => {
            db.close();
            console.log(result);
            callback(err,result);
         });
      });
   }

   /*
   * Name of the Method : getMessages
   * Description : To fetch messages from DB between two users.
   * Parameter :
   *      1) userID
   *      2) callback function
   * Return : callback
   */
   logout(userID,isSocketId,callback){

      const data = {
         $set :{
            online : 'N'
         }
      };
      this.Mongodb.onConnect( (db,ObjectID) => {

         let condition = {};
         if (isSocketId) {
            condition.socketId = userID;
         }else{
            condition._id = ObjectID(userID);
         }


         db.collection('users').update( condition, data ,(err, result) => {
            db.close();
            callback(err,result.result);
         });
      });
   }

   /*
   * Name of the Method : isUserLoggedOut
   * Description : Checks weather user is online or offile.
   * Parameter :
   *      1) userSocketID
   *      2) callback function
   * Return : callback
   */
   isUserLoggedOut(userSocketId,callback){
      this.Mongodb.onConnect( (db,ObjectID) => {
         db.collection('users').findOne({ socketId: userSocketId},(error, result) => {
            db.close();
            if (error) {
               callback({loggedOut:true});
            }else{
               if (result===null) {
                  callback({loggedOut:true});
               }else{
                  if (result.online === 'Y') {
                     callback({loggedOut:false});
                  }else{
                     callback({loggedOut:true});
                  }
               }
            }
         });
      });
   }
}

module.exports = new Helper();
```

CHAPTER 6. CREATING NEW ANGULAR APPLICATION

Now, we will start the implementation of Angular application. Here we will use Angular’s some the most useful features also we will use some third party plugin to enhance the look and feel of the our application.

In this application, we will use Angular Routing, Angular services and we will create a Models for server side response in order to get a proper and expected response from server.

Here, we will use **Angular bootstrap** in order to use tabs and Modals and to show the aesthetic notification, I will use **ng2-toaster** module.

Here we will use Angular CLI to create a new Angular project. At the time of implementing this application, I had installed **1.0.0 version** Angular CLI and **6.10.0 version** Nodejs. However, I will update this application with any new release of the Above tool.

But first let’s see the list of features that we are going to implement in this chat application.

FEATURES:

1. Login and registration.
2. Users can message each other.
3. Maintaining a chat list of users even if all the users are offline.
4. We will show the conversation between two users.
5. Feature to start a new conversation with a new user.
6. We will implement a feature to show the Notification, if a user goes offline OR a user appears online.
7. Also we will implement a Notification system when user receives a new Message.

We will cover each of these above mentioned features in below chapters down the road.

INSTALLING ANGULAR CLI

As we all know, we will use Angular CLI to create new Angular application.

WHY ANGULAR CLI?

Yeah only 4 words, *it makes everything easy*. Let's face it Angular CLI is a great tool while writing angular applications due to many reasons. For example, you need not to setup the project bundle, Angular CLI will take care of that. You can create components, services etc. If you don't have Angular CLI installed on your machine run the below command and install it globally.

```bash
BASH

npm install -g angular-cli
```

So, now you have Angular CLI installed so let's use Angular CLI to setup our application. After Angular CLI installation, to create a new Angular Project Run below command.

```bash
BASH

ng new AppName
```

This command will create all the necessary files, download all the required external dependencies and do all of the setups work for us. Now run your application by using below command and check the output at **localhost:4200** in the browser.

```bash
BASH

ng serve
```

APPLICATION DIRECTORY STRUCTURE

We have already created our Angular application with the help of Angular CLI. Now it's time to create components and services for our application but before that let's take a look at application directory structure, in order to know where each file resides.

When you open your application directory you will see there are three folders **/e2e**, **/node_modules** and **/src**. Now open **/src** folder you will find, there is a folder named as **/app**. We will create our files inside the **/app** folder.

Directory Structure

```
+---/auth
|
|   +-- auth.component.ts
|   +-- auth.component.html
|   +-- auth.component.css
|
+---/conversation
|
|   +-- conversation.component.ts
|   +-- conversation.component.html
|   +-- conversation.component.css
|
+---/home
|
|   +-- home.component.ts
|   +-- home.component.html
|   +-- home.component.css
|
+---/models
|
|   +-- auth.ts
|   +-- chat-list.ts
|   +-- common.ts
|   +-- conversation.ts
|   +-- session-check.ts
|
+---/new-chat
|
|   +-- new-chat.component.ts
|   +-- new-chat.component.html
|   +-- new-chat.component.css
|
+---/not-found
|
|   +-- not-found.component.ts
|   +-- not-found.component.html
|
+---/services
|
|   +-- app.service.ts
|   +-- emitter.service.ts
|   +-- http.service.ts
|   +-- socket.service.ts
|
+-- app.component.css
+-- app.component.html
+-- app.component.ts
+-- app.module.ts
+-- app.routing.ts
```

CREATING A MODEL FOR OUR APPLICATION

We have already seen the server side response of endpoints and socket events. Now, let’s create a models for Angular project listed below.

AUTH:

As the name suggests, this model will be used in authentication process. Now create auth.js file and write down the below code.

AUTH.TS:

```typescript
export class Auth {
  constructor(
        public error: boolean,
        public message: String,
        public userId: String
  ) {}
}
```

CHAT-LIST:

Create chat-list.js and write below code. This model beautifies the chat list response coming from the server.

CHAT-LIST.TS:

```typescript
export class ChatList {
  constructor(
       public _id: string,
       public online: string,
       public socketId: string,
       public timestamp :number,
       public username :string
  ) {}
}
```

COMMON:

Am using this model for any random response coming from server. Create a **common.js** and write below down the below code into it.

COMMON.TS:

```typescript
export class Common {
  constructor(
     public error: boolean,
     public message: String,
  ) {}
}
```

CONVERSATION:

When server sends the response for conversation between two users, then this models comes into the rescue. Create a conversation.js and write down below code.

CONVERSATION.TS:

```typescript
export class Conversation {
  constructor(
    public message: String,
    public fromUserId: String,
    public toUserId: String,
    public timestamp:Number,
    public username:String
  ) {}

}
```

SESSION-CHECK:

When user logs out from the application then we will use this model. Create session-check.js file inside models folder and write down below code.

SESSION-CHECK.TS:

```typescript
export class sessionCheck {
  constructor(
      public error: boolean,
      public username: String,
      public message: String
  ) {}

}
```

CHAPTER 7. CREATING APPLICATION SERVICES

In this chapter, we will create services required for our application. Basically I have created 4 services, inside the **/services** folder which are listed below,

1. emitter.service
2. http.service
3. app.service
4. socket.service

In this application we will write all the logic and heavy server side operations inside these services. So let's see each services one by one.

EMITTER.SERVICE:

Am using this services for communication between components. I found this from here.

EMITTER.SERVICE.TS:

TYPESCRIPT

```
import {Injectable, EventEmitter} from '@angular/core';

@Injectable()

export class EmitterService {
    private static _emitters: { [ID: string]: EventEmitter<any> } = {};
    static get(ID: string): EventEmitter<any> {
      if (!this._emitters[ID]){
                    this._emitters[ID] = new EventEmitter();
      }
      return this._emitters[ID];
    }

}
```

HTTP.SERVICE:

In this service, I will use Angular's http module to call the API for our chat application. In this Service I have created 6 methods userNameCheck(), login(), registerUser(), getMessages() and getUsersToChat().

hTTP.SERVICES.TS:

```
/*
* Real time private chatting app using Angular,Nodejs, mongodb and Socket.io
* @author Shashank Tiwari
*/

/* Importing from core library starts*/
import { Injectable } from '@angular/core';
import { Http, Response, Headers, RequestOptions } from '@angular/http';
/* Importing from core library ends*/

/* Importing from rxjs library starts*/
import { Observable } from 'rxjs/Rx';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';
/* Importing from rxjs library ends*/

/* Importing Models starts */
import { Common } from './../models/common';
import { Auth } from './../models/auth';
import { sessionCheck } from './../models/session-check';
import { Conversation } from './../models/conversation';
import { ChatList } from './../models/chat-list';
/* Importing Models ends */

@Injectable()
export class HttpService {

   /*
   * specifying Base URL.
   */
   private BASE_URL = 'http://localhost:4000/';

   /*
   * Setting the Request headers.
   */
   private headerOptions = new RequestOptions({
      headers : new Headers({ 'Content-Type' : 'application/json;charset=UTF-8' })
   });

   constructor( private http:Http) { }

   public userNameCheck(params){
     return this.http.post('${this.BASE_URL}usernameCheck',JSON.stringify(params),this.headerOptions)
     .map( (response:Response) => {
         let data = response.json();
         return new Common(data.error,data.message)
     })
     .catch( (error:any) => Observable.throw(error.json().error || 'Server error') );
   }

   public login(params){
     return this.http.post('${this.BASE_URL}login',JSON.stringify(params),this.headerOptions)
                      .map( (response:Response) => {
            let data = response.json();
            return new Auth(data.error,data.message,data.userId)
         })
                      .catch( (error:any) => Observable.throw(error.json().error || 'Server error') );
   }

   public registerUser(params){
     return
     this.http.post('${this.BASE_URL}registerUser',JSON.stringify(params),this.headerOptions)
                      .map( (response:Response) => {
            let data = response.json();
            return new Auth(data.error,data.message,data.userId)
         })
         .catch( (error:any) => Observable.throw(error.json().error || 'Server error') );
   }

   public userSessionCheck(params){
      return this.http.post('${this.BASE_URL}userSessionCheck',JSON.stringify(params),this.headerOptions)
      .map( (response:Response) => {
           let data = response.json();
           return new sessionCheck(data.error,data.username,data.message)
      })
      .catch( (error:any) => Observable.throw(error.json().error || 'Server error') );
```

```
  }

  public getMessages(params){
    return this.http.post('${this.BASE_URL}getMessages',JSON.stringify(params),this.headerOptions)
      .map( (response:Response) => {
        let data = response.json();
        let newMessages = data.messages.map(message => {
          return new Conversation(
            message.message,
            message.fromUserId,
            message.ToUserId,
            message.timestamp,
            message.username,
          );
        });
        data.messages = newMessages;
        return data;
      })
      .catch( (error:any) => Observable.throw(error.json().error || 'Server error') );
  }

  public getUsersToChat(params){
    return this.http.post('${this.BASE_URL}getUsersToChat',JSON.stringify(params),this.headerOptions)
      .map( (response:Response) =>{
        let data = response.json();
        let chatList = data.chatList.map(list => {
          return new ChatList(
              list._id,
              list.online,
              list.socketId,
              list.timestamp,
              list.username
            )
          });
        data.chatList = chatList;
        return data;
      })
      .catch( (error:any) => Observable.throw(error.json().error || 'Server error') );
  }

}
```

Explanation:

Here I am using Observables, **rxjs library** and I have imported them. After that to get expected am using the models that we just created above.

In this class, we will define five methods which calls the API endpoints to the server and at the same point they consume the models to frame the expected server response for the chat application.

SOCKET.SERVICE.TS:

With the help of this service our application will receive real-time notifications and can send real-time updates to the server. In this service, I am using **socket.io-client**. This module helps us to communicate with the socket server. To install this module run the below command.

```bash
npm install @types/socket.io-client --save
```

Here in this class, we will define five methods listed below. Each of these method has own uses and we will consume these methods from Angular components.

1. connectSocket()
2. sendMessage()
3. logout()
4. receiveMessages()
5. getChatList()

SOCKET.SERVICE.TS:

```typescript
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs/Subject';
import { Observable } from 'rxjs/Observable';

import * as io from 'socket.io-client';

/* Importing Models starts */
import { ChatList } from './../models/chat-list';
import { Conversation } from './../models/conversation';
/* Importing Models ends */

@Injectable()
export class SocketService {

    /*
    * specifying Base URL.
    */
    private BASE_URL = 'http://localhost:4000';
    public socket;

    constructor() {}
    /*
    * Method to connect the users to socket
    */
    connectSocket(userId:string){
                    this.socket = io(this.BASE_URL,{ query: `userId=${userId}`});
    }

    /*
    * Method to emit the add-messages event.
    */
    sendMessage(message:any):void{
                    this.socket.emit('add-message', message);
    }

    /*
    * Method to emit the logout event.
    */
    logout(userId):any{

                    this.socket.emit('logout', userId);

                    let observable = new Observable(observer => {
                       this.socket.on('logout-response', (data) => {
                           observer.next(data);
                       });
            return () => {
                           this.socket.disconnect();
                       };
                    })
                    return observable;
    }

    /*
    * Method to receive add-message-response event.
    */
    receiveMessages():any{
                    let observable = new Observable(observer => {
                    this.socket.on('add-message-response', (data) => {
                         observer.next(
                                               new Conversation(
                                                   data.message,
                                                   data.fromUserId,
                                                   data.toUserId,
                                                   data.timestamp,
                                                   data.username
                                               )
                           );
                        });
                        return () => {
                                             this.socket.disconnect();
                        };
                    });
                    return observable;
    }

    /*
    * Method to receive chat-list-response event.
    */
    getChatList(userId:string):any {

                    this.socket.emit('chat-list' , { userId : userId });

                    let observable = new Observable(observer => {
                        this.socket.on('chat-list-response', (data) => {
                            if(data.chatList !== null && data.chatList !== undefined) {
                            let userChatList = data.chatList;
                            let modelChatList = [];
                            if(userChatList !== null) {
                                for (var i = 0; i < userChatList.length; i++) {
                                    modelChatList.push(
                                                                new ChatList(
                                                                    userChatList[i]._id,
                                                                    userChatList[i].online,
                                                                    userChatList[i].socketId,
                                                                    userChatList[i].timestamp,
                                                                    userChatList[i].username
                                                                )
                                                   );
                                               }
                                delete data.chatList;
                    data.chatList = modelChatList;
                }
                        }
                        observer.next(data);
                    });
                    return () => {
                       this.socket.disconnect();
                    };
     })
     return observable;
    }

}
```

APP.SERVICE:

In this service, we will consume **httpService** and we will call API endpoint. In this service I have created six methods listed below,

1. userNameCheck()
2. login()
3. registerUser()
4. userSessionCheck()
5. getMessages()
6. getUsersToChat()

This service is consumed by the all the components for operation such as Login, registration and fetching messages through the API endpoint.

APP.SERVICE.TS:

```typescript
TYPESCRIPT

import { Injectable } from '@angular/core';

/* Importing http service starts*/
import { HttpService } from './http.service';
/* Importing http service ends*/

/* Importing Models starts */
import { Common } from './../models/common';
import { Auth } from './../models/auth';
import { sessionCheck } from './../models/session-check';
import { Conversation } from './../models/conversation';
/* Importing Models ends */

@Injectable()
export class AppService {

    constructor(public httpService : HttpService) {
    }

    /*
    * check if username already exists.
    */
    public userNameCheck(params,callback){
                    this.httpService.userNameCheck(params).subscribe(
                        response => {
                callback(response);
                        },
                        error => {
                                        callback(new Common( true,'Error occured,Please try after some time.'));
                        }
                    );
    }

    /*
    * Login the user
    */
    public login(params ,callback):any{
                    this.httpService.login(params).subscribe(
                        response => {
                                        callback(response);
                        },
                        error => {
                                        callback(new Auth(true,'Error occured,Please try after some time.',null));
                        }
                        );
    }

    /*
    * method to add new users
    */
    public registerUser(params,callback):any{
                        this.httpService.registerUser(params).subscribe(
                            response => {
                        callback(response);
                            },
                            error => {
                                        callback(new Auth(true,'Error occured,Please try after some time.',null));
                            }
                        );
   }

   /*
   * Method to check the session of user.
   */
   public userSessionCheck(userId , callback):any{
                        this.httpService.userSessionCheck({userId : userId}).subscribe(
                response => {
                        callback(response);
                },
                error => {
                        callback(new sessionCheck(true,null,'You not loogged in.'));
                }
        );
   }


   /*
   * method to get the messages between two users
   */
    public getMessages(params ,callback):any{
                        this.httpService.getMessages(params).subscribe(
                response => {
                        callback(false,response);
                },
                            error => {
                        callback(true,'HTTP fail.');
                }
        );
    }

    /*
    * method to get the users to chat (start a new chat)
    */
    public getUsersToChat( params,callback):any{
                        this.httpService.getUsersToChat(params).subscribe(
                           response => {
                        callback(false,response);
              },
              error => {
                        callback(true,'HTTP fail.');
              }
        );
    }


 }
```

CHAPTER 8. COMPONENTS, ROUTER AND APP.MODULE

In this chapter, we will focus on **app.module.ts** file, And we will create all the components which are required for the application. Here I am using **ng2-toastr** and **ng2-bootstrap** module to increase the user experience so we will install these modules also. As our Angular application is a single page application, so here I will create a routing for our application.

COMPONENTS

So let's first create all the components and understand usage of them. In this application I will create 6 components and If I include app component which ships with the angular application when you create a new application using Angular Cli, so total we have 7 components.

To create new component, use the below command. The below command will create a new component. And below is the list of all the components with explanation.

```bash
BASH

ng g component my-new-component
```

APP.COMPONENT:

As I told earlier this component by default comes with the Angular application when we create new app with Angular CLI and located inside /app folder.

HOME.COMPONENT:

In the home component, we will show list of users, entire conversation between two users, Button to start a new chat (which eventually will open a model with list of users with whom you can start a new conversation) and logout button, so I will use all the components. Lot of things in a single component? No never, we will add other component's selector into this component and our **home.component.ts** will be sweet,short and clean. This component is located inside /app/home directory.

AUTH.COMPONENT:

As the Name suggests, in this component will hold code regarding login and Registration of the users. This component is located inside /app/auth directory.

CHATLIST.COMPONENT:

In this component, I will show the list of chats that logged in user have. When logged in user selects any user from the list I will show the entire conversation between both the users with the help of conversation component.

This component is located inside /app/chat-list directory.

CONVERSATION.COMPONENT:

In this component, we will render the conversation between two users and This component is located inside /app/conversation directory.

NEWCHAT.COMPONENT:

When user clicks on the start new chat button I will use this component. In this component I am showing the list of user to start a new chat. Here logged in user can select a user with whom he/she wants to start a new chat from the od user. This component is located inside /app/new-chat directory.

NOTFOUND.COMPONENT:

When user enters any non-existing URL manually then we will show this component. This component is located inside /app/not-found directory.

Now we have created all the components now let's jump on to the application Routing.

ROUTER

Now create **app.routing.ts** file inside the /app folder and write down below code.

APP.ROUTING.TS:

```typescript
import { ModuleWithProviders } from '@angular/core';

import { Routes , RouterModule } from '@angular/router';

import { AuthComponent } from './auth/auth.component';
import { HomeComponent } from './home/home.component';
import { NotFoundComponent } from './not-found/not-found.component';


const appRoutes :Routes = [
   { path : '' , component : AuthComponent},
   { path : 'home' , component : HomeComponent},
   { path : 'home/:userid' , component : HomeComponent},
   { path : '**' , component : NotFoundComponent},
];

export const appRouting :ModuleWithProviders = RouterModule.forRoot(appRoutes);
```

Explanation:

In the above code, we have imported our components, Routes and RouterModule. With the help of Angular's powerful routing, we have defined the appRoutes.

Here I have defined two routes **/home** and **/home/:userid** respectively. And if url is not found by angular app then we will use **NotFoundComponet**.

Now open **app.component.html** file where our angular router will render the component templates. So to make this happen we will use **router-outlet**.

Note: You can remove the app.component if you want and instead of that you can directly use index.html file.

APP.COMPONENT.HTML:

```html
<router-outlet></router-outlet>
```

Now we have completed the routing for our application. Now let’s import all the components and third party module and routing in our **app.module.ts** file.

APP.MODULE

Here I will import all the files, module and Router as well. Now open app.module.ts file, Write below code.

APP.MODULE.TS

```typescript
TYPESCRIPT
/* Importing from core library starts*/
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
/* Importing from core library ends*/
/* Importing from bootstrap library starts*/
import { ModalModule } from 'ng2-bootstrap';
import { TabsModule } from 'ng2-bootstrap';
import { ToastModule } from 'ng2-toastr/ng2-toastr';
/* Importing from bootstrap library ends*/
/* Importing application routing starts*/
import { appRouting } from './app.routing';
/* Importing application routing ends*/
/* Importing Application services starts*/
import { AppService } from './services/app.service';
import { HttpService } from './services/http.service';
import { SocketService } from './services/socket.service';
import { EmitterService } from './services/emitter.service';
/* Importing Application services ends*/
/* Importing application components starts*/
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { AuthComponent } from './auth/auth.component';
import { ChatListComponent } from './chat-list/chat-list.component';
import { ConversationComponent } from './conversation/conversation.component';
import { NewChatComponent } from './new-chat/new-chat.component';
import { NotFoundComponent } from './not-found/not-found.component';
/* Importing application components ends*/

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    AuthComponent,
    ChatListComponent,
    ConversationComponent,
    NotFoundComponent,
    NewChatComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    appRouting,
    TabsModule.forRoot(),
    ModalModule.forRoot(),
    ToastModule.forRoot()
  ],
  providers : [HttpService,SocketService,AppService,EmitterService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

The above code is self-explanatory so Here I won't talk much about it. Here I have added tabs and modal module from the ng2-bootstrap module.

CHAPTER 10. CREATING LOGIN AND REGISTRATION PAGE

In this Chapter, we will create login and Registration functionality for our application. To perform login and Registration operation will consume **appService** as well as **httpService**. So basically we will work on auth component, app service, and Http service, not much work to do in this chapter. Alright, here I will be using ng-bootstrap for Angular 2 in this application to implement UI.

So take a look at **ng-bootstrap** for Angular 2.

STARTING WITH HTML

First, we will create registration part in addition to this we will add functionality to check the availability of username. And then, we will implement the login page for this application. So enough talking let's get our hands dirty open the **auth.component.html** and add the below HTML.

AUTH.COMPONENT.HTML:

HTML

```html
<div class="container app-screen">

  <p class="app-heading" >
    <span>Realtime private Chatting application Version 3 </span>
    <span class="app-heading-tech">using Angular 2 and Nodejs</span>
  </p>

  <div class="row login-screen">
    <div class="col-md-offset-3 col-md-5 user-registration justifier">
      <tabset>
        <!-- Login box starts -->
        <tab heading='Login'>
          <div class="justifier">
            <div class="form-group">
              <label>Username</label>
              <input type="text"
                name="username"
                class="form-control"
                placeholder="Enter your Email"
                [(ngModel)]="username"
                value="{{username}}"
              >
            </div>

            <div class="form-group">
              <label>Password</label>
              <input type="password"
                name="password"
                class="form-control"
                placeholder="Enter your Password"
                [(ngModel)]="password"
                value="{{ password }}"
              >
            </div>
            <div class="form-group">
              <button class="btn submit-btn"  (click)="login()">Login</button>
            </div>
          </div>
        </tab>
        <!-- Login box ends -->
        <tab heading='Registration'>
          <!-- Registration box starts -->
          <div class="justifier">

            <div class="form-group">
              <label>Name</label>
              <input type="text"
                name="name"
                class="form-control"
                placeholder="Enter your Name"
                [(ngModel)]="username"
                value="{{username}}"
                (keyup)="onkeyup($event)"
                (keydown)="onkeydown($event)"
              >
              <br *ngIf="isuserNameAvailable && username!=''" />
              <div *ngIf="isuserNameAvailable && username!=''" class="alert alert-danger">
                username <strong>{{username}}</strong> is already taken.
              </div>
            </div>

            <div class="form-group">
              <label>Email</label>
              <input type="text"
                name="email"
                class="form-control"
                placeholder="Enter your Email"
                [(ngModel)]="email"
                value="{{email}}"
              >
            </div>

            <div class="form-group">
              <label>Password</label>
              <input type="password"
                name="password"
                class="form-control"
                placeholder="Enter your Password"
                [(ngModel)]="password"
                value="{{ password }}"
              >
            </div>

            <div class="form-group">
              <button class="btn submit-btn" (click)="registerUser()">Registration</button>
            </div>
          </div>
          <!-- Registration box ends -->
        </tab>
      </tabset>
    </div>
  </div>

</div>
```

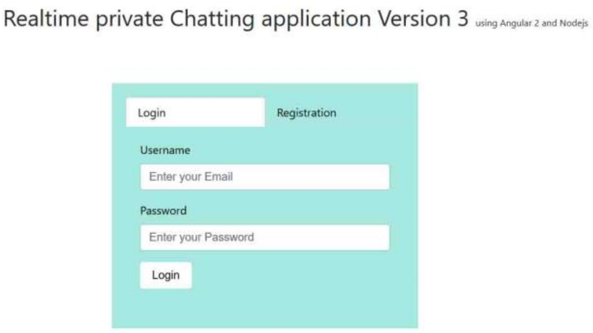
Explanation:

The above code is sweet and simple here we are `<ngb-tabset>` and `<ngb-tab>` so we can achieve the Tab set in our application. So let's get started with login and after that, we will come to Registration.

Inside the `<ngb-tab title="Login">`, we have two input tags, where the user would write his/her username and password after doing that hit login button to see the home page.

Inside the `<ngb-tab title="Registration">`, we have three inputs tags to enter Username, email and password and button to register user the new user.

And everything looks self-explanatory, isn't it? Now if you open application you should see something like below,


Realtime private Chatting application Version 3 using Angular 2 and Nodejs
Login
Registration
Username
Enter your Email
Password
Enter your Password
Login

THE .TS FILE (COMPONENT):

Now let's write down the logic behind the auth component, So for that open **auth.component.ts** file and write the below code. In the below code, we are performing all the operations i.e. login, username availability checks, and Registration of the new user.

AUTH.COMPONENT.TS:

TYPESCRIPT

```typescript
/* Importing from core library starts*/
import { Component,ViewContainerRef } from '@angular/core';
import { Router } from '@angular/router';
/* Importing from core library ends*/

/* Importing ToastsManager library starts*/
import { ToastsManager } from 'ng2-toastr/ng2-toastr';
/* Importing ToastsManager library ends*/

/* Importing Application services starts*/
import { AppService } from './../services/app.service';
import { HttpService } from './../services/http.service';
```

```
/* Importing Application services ends*/

@Component({
  selector: 'app-auth',
  templateUrl: './auth.component.html',
  styleUrls: ['./auth.component.css'],
})

export class AuthComponent{

  /*
  * Variables to host data for this component starts
  */

  /* Variables to hold users data (ng-model) for this component starts*/
  private username = null;
  private email = null;
  private password = null;

  /* Will be used to hide and show the errors for username availability check*/
  private isuserNameAvailable = false;

  /* Used for username availability check funcnality*/
  private userTypingTimeout= 500;
  private typingTimer = null;
  /*
  * Variables to host data for this component ends
  */

  constructor(
      private toastr: ToastsManager,
      private _vcr: ViewContainerRef,
      private chatService : AppService,
      private router :Router
    ) {
    this.toastr.setRootViewContainerRef(_vcr);
  }


  /*
  * Method to check the availability of the username starts
  */
  public onkeyup(event){

    clearTimeout(this.typingTimer);

    this.typingTimer = setTimeout( ()=>{

      this.chatService.userNameCheck({
        'username' : this.username
      }, (response)=>{
        if(response.error) {
          this.isuserNameAvailable = true;
        }else{
          this.isuserNameAvailable = false;
        }
      });

    }, this.userTypingTimeout);
  }

  public onkeydown(event){
    clearTimeout(this.typingTimer);
  }
  /*
  * Method to check the availability of the username ends
  */

  /*
  * Method to Login the new user starts
  */
  public login():void{
    if(this.username === '' || this.username === null) {
      this.toastr.error("Username can't be empty.", 'Fill the form');
    }else if(this.password === '' || this.password === null ){
      this.toastr.error("Password can't be empty.", 'Fill the form');
    }else{
        this.chatService.login({
            'username' : this.username,
            'password' : this.password,
        },(response)=>{
          if(!response.error) {
              this.router.navigate(['/home/'+response.userId]);
          }else{
              this.toastr.error(response.message, 'No trespassing.');
          }
        });
      }
    }
    /*
    * Method to Login the new user starts
    */



    /*
    * Method to register the new user starts
    */
    public registerUser():void{

      if(this.username === '') {
        this.toastr.error("Username can't be empty.", 'Fill the form');
      }else if(this.email === ''){
        this.toastr.error("Email can't be empty.", 'Fill the form');
      }else if(this.password === ''){
        this.toastr.error("Password can't be empty.", 'Fill the form');
      }else{
        this.chatService.registerUser({
            username : this.username,
            email : this.email,
            password : this.password
        },(response)=>{
            if(!response.error) {
              this.router.navigate(['/home/'+response.userId]);
            }else{
              this.toastr.error(response.message, 'This very Rear.');
            }
        });
      }
    }
    /*
    * Method to register the new user ends
    */
}
```

Explanation:

Starting from imports, first, we have imported component and router (for single page application). Next, we have imported services i.e. App Service and HTTP service in order to consume them. Now Inside the component decorator, in providers array add our services.

As you can see the class AuthComponent contains the two primary methods login() and registration().

But before that first let's take a look at **username check availability**, In AuthComponent class you might have noticed that we have onekeyup() and onkeydown() method. The idea here to send an HTTP request to get the availability of username when the user finishes the typing in the input box.

When the user finishes the typing we will call the userNameCheck() method which defined inside the **appService.ts**.

In the login function, we are calling the login() method along with username and password, which is defined inside the App Service.

Where as in registration function, we calling the registration(), which is again defined inside the App Service and requires the username email and password as parameters.

THE ROUTER:

After the successful login or after registration of the new user, we will redirect the user to the home along with **userId** as URL parameter. So do that I have taken the advantage of Angular's router.

In the below code, I have used the navigate method of the angular router where you would give the URL, which should be defined in **appRouting.ts** file which we created in the previous part.

AUTH.COMPONENT.TS:

```typescript
this.router.navigate(['/home/'+response.userId]);
```

CHAPTER 11. IMPLEMENTING HOME PAGE

In this chapter, we will implement the home page. In this chapter I will implement the chat list, functionality to start a new chat, to send message and receive messages in real time, along with notification and other important things.

And the end we will combine each of the module. Here we will implement four key features into our application.

First List of online users, with whom the user can chat. Second real-time private chat between two users. Third list of users with whom logged in user can start a new chat and at the end we will show the notification on the arrival of any new user.

So let's start, we will complete our home page by implementing below listed points,

1. *We will check session of the user (on page refresh) and Logout functionality.*
2. *Getting the list of online users and we will show the notification on appearance of a new user.*
3. *We will Select a user to chat with and retrieve the conversation between them.*
4. *Sending a new message.*
5. *We will add a button and on click of that we will show the list of users with whom a logged in user can start a new chat.*

1. CHECKING SESSION OF THE USER AND LOGGING OUT

Open **home.component.ts** file, and below code. In the below code, we will check the session of the user by making an HTTP request to the server.

HOME.COMPONENT.TS:

```typescript
TYPESCRIPT
/* Importing from core library starts*/
import { Component, OnInit, Input, OnChanges, ViewChild, ViewContainerRef } from '@angular/core';
import { ActivatedRoute,Router } from '@angular/router';
/* Importing from core library ends*/

import { ChatListComponent } from '../chat-list/chat-list.component';
import { ConversationComponent } from '../conversation/conversation.component';
import { NewChatComponent } from '../new-chat/new-chat.component';
```

```
/* Importing ToastsManager library starts*/
import { ToastsManager } from 'ng2-toastr/ng2-toastr';
import { ModalDirective } from 'ng2-bootstrap';
/* Importing ToastsManager library ends*/

/* Importing Application service(i.e. AppService) and http service starts*/
import { AppService } from './../services/app.service';
import { HttpService } from './../services/http.service';
import { SocketService } from './../services/socket.service';
import { EmitterService } from './../services/emitter.service';
/* Importing Application service(i.e. AppService) and http service ends*/

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css'],
})
export class HomeComponent implements OnInit {

  /*
  * Chat and message related variables starts
  */
  private userId = null;
  private username = null;
  private userSocketId =null;
  /*
  * Chat and message related variables ends
  */

  @ViewChild(ChatListComponent) chatListComponent: ChatListComponent
  @ViewChild(ConversationComponent) conversationComponent: ConversationComponent
  @ViewChild(NewChatComponent) NewChatComponent: NewChatComponent
  @ViewChild('lgModal') public lgModal: ModalDirective;

  private conversation = 'CONVERSATION';

  constructor(
    private socketService : SocketService,
    private httpService : HttpService,
    private appService : AppService,
    private route :ActivatedRoute,
    private router :Router,
    private _emitterService: EmitterService
  ) { }

  ngOnInit() {
    /*
    * getting userID from URL using 'route.snapshot'
    */
    this.userId = this.route.snapshot.params['userid'];
    if(this.userId === '' || typeof this.userId == 'undefined') {
      this.router.navigate(['/']);
    }else{
      /*
      * Checking if user is logged in or not, starts
      */
      this.appService.userSessionCheck(this.userId,( response )=>{
        this.username = response.username;
        /*
        *   -------------------
        *  Here we will make socket connection and we will fetch chat list
        *   -------------------
        *   ------------------
        *   Here we will listen for any new incoming messages
        *   -------------------
        */
      });
      /*
      * Checking if user is logged in or not, ends
      */
    }
  }

  private logout(){
    this.socketService.logout({userId : this.userId}).subscribe(response => {
      this.router.navigate(['/']); /* Home page redirection */
```

```
        });
    }
}
```

Explanation:

Let's first start from imports, I have done import of **Component**, **OnInit**, **Input**, **OnChanges**, **ViewChild** and **ViewContainerRef**.

Here I will talk about **ViewChild, ViewContainerRef** and rest of the modules are not that much interesting.

Here with the help of **ViewChild**, I can call methods defined in other component which I will use in home component. **ViewContainerRef** Is useful for toaster module. And now rest of the imports are self-explanatory.

As you can see in above code, the HomeComponent class implemets ngOnInit() method after that we have some variables to hold the data.

Inside the ngOnInit(), we are capturing the **userId** of the user from the Url by using the **snapshot interface**. If in any case **userId** found null or blank then on the next line we will redirect the user to login page.

Or else in the ideal situation, we will check the session by passing the **userId** of the respective user to userSessionCheck() which is defined inside the appService. If everything goes fine, then we will connect the user to the socket.

And at the end, in logout() method, we will log out the user by sending an event to the server.

2. THE LIST OF ONLINE USERS

Let's get the list of online users, to do that we will use socket service. The code to get the list of users goes inside the userSessionCheck() method for an obvious reason. So how do we do that, take look at below code,

HOME.COMPONENET.TS:

```typescript
TYPESCRIPT
export class HomeComponent implements OnInit {

  /*
  * ------------------------
  *  Some variables
  * -------------------------
  */
  constructor(
    private socketService : SocketService,
    private httpService : HttpService,
    private appService : AppService,
    private route :ActivatedRoute,
    private router :Router,
    private _emitterService: EmitterService
  ) { }

  ngOnInit() {
    /*
    * getting userID from URL using 'route.snapshot'
    */
    this.userId = this.route.snapshot.params['userid'];
    if(this.userId === '' || typeof this.userId == 'undefined') {
      this.router.navigate(['/']);
    }else{
      /*
      * Checking if user is logged in or not, starts
      */
      this.appService.userSessionCheck(this.userId,( response )=>{
        this.username = response.username;

        / *
        * Making socket connection by passing UserId.
        */
        this.socketService.connectSocket(this.userId);

        / *
        * Calling function to get the chatlist.
        */
        this.chatListComponent.getChatList();

        / *  ------------------
         *   Here we will listen for any new incoming messages
         *   --------------------
         */
      });
      /*
      * Checking if user is logged in or not, ends
      */
    }
  }
```

Explanation:

The below code connects the client to the socket server, along with the connection here we are passing user id of the client.

```typescript
this.socketService.connectSocket(this.userId);
```

connectSocket() is defined inside the socket Service. Now that we have connected the client to socket now let's fetch the chat list. As you can see the we have called the getChatList() method which defined inside the **chatListComponent**.

Now we will look into the chatListComponent, so for that we first I will start with markup. Open the **chat-list.component.html** and write down below code.

CHAT-LIST.COMPONENT.HTML:

```html
HTML
<div class="user-list-wrapper">
  <ul class="user-list">

    <!-- click event to handle the selection : selectedUser() -->
    <!-- adding a class to indicate the selection -->
    <li *ngFor="let user of chatListUsers"
      (click)="selectedUser(user)"
      [class.selected-user]="isUserSelected(user._id)"
      >
        {{ user.username}}
      <span *ngIf="user.online=='Y'" class="online-user" title="Online"></span>
      <span *ngIf="user.online=='N'" class="offline-user" title="Offline"></span>
    </li>
  </ul>
</div>
```

The code written inside the chat-list.component.html is ridiculously easy to understand So I won’t talk about it.

Now let’s take a look at the chat-list.component.ts, where I have written all the logic behind chat list and user appearance notification. Now open **chat-list.component.ts** and write down below code.

CHAT-LIST.COMPONENT.TS:

```typescript
TYPESCRIPT
/* Importing from core library ends*/
import { Component, OnInit, ViewContainerRef,Input } from '@angular/core';
import { Router,ActivatedRoute } from '@angular/router';
/* Importing from core library ends*/

/* Importing ToastsManager library starts*/
import { ToastsManager } from 'ng2-toastr/ng2-toastr';
/* Importing ToastsManager library ends*/

/* Importing Application services starts*/
import { AppService } from './../services/app.service';
import { HttpService } from './../services/http.service';
import { SocketService } from './../services/socket.service';
import { EmitterService } from './../services/emitter.service';
/* Importing Application services ends*/
import { ChatList } from './../models/chat-list';

@Component({
  selector: 'app-chat-list',
  templateUrl: './chat-list.component.html',
  styleUrls: ['./chat-list.component.css']
})
export class ChatListComponent implements OnInit {

  /*
  * Variables to host data for this component starts
  */
  private userId:string = null;
  private selectedUserId:string =  null;
  private selectedUserName:string = null;
  private chatListUsers:ChatList[] = [];
  /*
  * Variables to host data for this component ends
  */
```

```
/*
* Incoming data from other component starts
*/
@Input() conversation: any;
@Input() selectedUserInfo:any;
/*
* Incoming data from other component ends
*/

constructor(
   private toastr: ToastsManager,
   private _vcr: ViewContainerRef,
   private route :ActivatedRoute,
   private router :Router,
   private appService : AppService,
   private socketService : SocketService
) {
   this.toastr.setRootViewContainerRef(_vcr);
}

/*
* Getting the userID from URL starts
*/
ngOnInit() {
   this.userId = this.route.snapshot.params['userid'];
}

getChatList():void{

   this.chatListUsers = [];

   if(this.userId  === '' || typeof this.userId === 'undefined') {
      this.toastr.error("Can't get the chat list,try after some time.", 'This is rear.');
   }else{
      /*
      * calling method of service to get the chat list.
      */
      this.socketService.getChatList(this.userId).subscribe(response => {
         if(!response.error) {

            if(response.singleUser) {

               /*
               * Removing duplicate user from chat list array.
               */
               if(this.chatListUsers !==null && this.chatListUsers.length > 0 ) {
                  this.chatListUsers = this.chatListUsers.filter( ( obj ) =>{
                     return obj._id !== response.chatList[0]._id;
                  });
               }

               /*
               * Adding new online user into chat list array
               */
               if(this.chatListUsers === null) {
                  this.chatListUsers= response.chatList;
               }else{
                  if(response.chatList.length > 0) {
                     this.chatListUsers.push(response.chatList[0]);
                  }
               }
               /*
               * Showing Notification when a new user comes online.
               */
               this.toastr.success(response.chatList[0].username+' appeared online.');

            }else if(response.userDisconnected){
               /*
               * Showing Notification when a user goes offline  starts.
               */
               if(this.chatListUsers !==null){
                  let offlineUser = null;
                  this.chatListUsers = this.chatListUsers.filter( ( obj ) => {
                     if(obj.socketId === response.socketId) {
                        offlineUser = obj;
                        obj.online = 'N';
                     }
```

```
                                return true;
                            });
                            if(offlineUser !== null) {
                                this.toastr.info(offlineUser .username+' Went offline.');
                            }
                        }
                        /*
                        * Showing Notification when a user goes offline ends.
                        */
                    }else{
                        /*
                        * Updating entire chatlist if user logs in.
                        */
                        this.chatListUsers = response.chatList;
                    }
                }else{
                    this.toastr.error("Can't get the chat list,try after some time.", 'This is rear.');
                }
            });
        }
    }

    /*
    * Method to select the user from the Chat list starts
    */
    private selectedUser(user):void{
        this.selectedUserId = user._id;
        this.selectedUserName = user.username;

        /*
        * Sending selected users information to other component starts.
        */
        EmitterService.get(this.selectedUserInfo).emit(user);
        /*
        * calling method to get the messages
        */
        this.appService.getMessages({ userId : this.userId,toUserId :user._id} , ( error , response)=>{
            /*
            * Sending conversation between two users to other component starts.
            */
            EmitterService.get(this.conversation).emit(response);
        });
    }

    /*
    * Method required for UI to indicate the selected user starts.
    */
    private isUserSelected(userId:string):boolean{
        if(!this.selectedUserId) {
            return false;
        }
        return this.selectedUserId ===  userId ? true : false;
    }
}
```

Explanation:

GETTING THE CHAT LIST:

The **ChatListComponent** class implementing the Oninit() method, where again I am capturing the user id from the Router. Now coming to the getChatList() method where chat list logic is handled.

In getChatList() method, we are calling a method from socket service which is callback function. Inside that call function basically we have if else loop. If you notice we have one if loop, one if else and lastly we have else loop. In the first if loop, I have written condition when some other user logs into the application.

if(response.singleUser), This indicates the some other user logged into in the application. And after that I am performing the check whether the incoming user is already into the chat list.

If yes, then update the list if user is not present into the chat list then I am adding that user into the chat list.

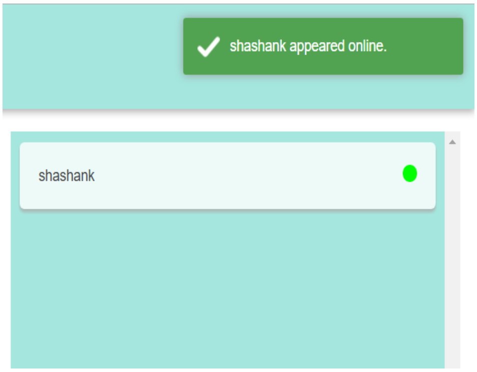
Now that we have added a new user into the chat list, so let's notify the logged into user that some has come online. So to do that we will show the notification.

Here I am using third party module **ng-toastr** to show the notification in Angular, which is very easy to setup and use.

You can read more about it, for example how to set up and use **ng-toastr** from here, as I am not going to talk about ng-toastr here in this E-book.

Or if chat list is completely empty, then we will definitely add the incoming new user with our both eye closed, isn't it?

And after all this addition of this new user, I am using toastr to show the notification that some new user appeared online as shown below.



Notification when a new user appears online.

In the second if else loop, I am removing an existing user from the chat list when that user goes offline. In the last else loop, I am crating the actual chat list.

FETCHING THE CONVERSATION BETWEEN TWO USER

Methods selectedUser() and isUserSelected() are used in the process of highlighting the select row from the chat list. But in the as you can see in the selectedUser() method, we are using getMessages() method which is defined inside the appService.ts file.

getMessages() method returns the conversation between two users, which is going to be used in the conversation component. So that we will render the conversation between two users.

So to send the conversation data from chatList component to conversation component I am taking help of **EmitterService**. In the next point I will cover the conversation component. How we will render the conversation between two users.

RENDRING THE CONVERSATION

Now that we have conversation data in the conversation component with the **EmitterService**. Let's take a look at Conversation component, how we will render the conversation between two users. Also, in this component we will provide functionality to send message from one user to another user.

Open the **conversation.component.html** and write down the below HTML. In the below code, I will just iterate the messages array.

CONVERSATION.COMPONENT.HTML:

```html
<div class="massege-wrapper">
  <div class="massege-container">

    <div *ngIf="selectedUserName"
      class="opposite-user">
      Chatting with {{selectedUserName}}
    </div>

    <ul class="message-thread">

      <li *ngFor="let message of messages"
```

```
            [class.align-right]="alignMessage(message.toUserId)"
            >
              {{ message.message}}
          </li>
        </ul>

    </div>
</div>
```

Now let’s see what happing inside the typescript of the conversation component. In the **conversation.component.ts** file, we will write code to receive the messages from other component as well as we will write code to receive messages from server in real time.

```
TYPESCRIPT
/* Importing from core library starts*/
import { Component, OnInit, Input, OnChanges,ViewContainerRef } from '@angular/core';
import { Router,ActivatedRoute } from '@angular/router';
/* Importing from core library ends*/

/* Importing ToastsManager library starts*/
import { ToastsManager } from 'ng2-toastr/ng2-toastr';
/* Importing ToastsManager library ends*/

/* Importing Application services starts*/
import { EmitterService } from './../services/emitter.service';
import { SocketService } from './../services/socket.service';
/* Importing Application services ends*/

@Component({
  selector: 'app-conversation',
  templateUrl: './conversation.component.html',
  styleUrls: ['./conversation.component.css'],
})
export class ConversationComponent implements OnInit {

  /*
  * Variables to host data for this component starts
  */
  private userId = null;
  private message = null;
  private selectedUser = null;
  private messages = null;
  /*
  * Variables to host data for this component ends
  */

  /*
  * Incoming data from other component starts
  */
  @Input() conversation: any;
  @Input() selectedUserInfo:any;
  /*
  * Incoming data from other component ends
  */

  constructor(
    private toastr: ToastsManager,
    private _vcr: ViewContainerRef,
    private route :ActivatedRoute,
    private router :Router,
    private socketService :SocketService
  ) { }

  ngOnInit() {
    this.userId = this.route.snapshot.params['userid'];
  }

  listenForMessages(){
    /*
```

```
        * subscribing for messages starts
        */
        this.socketService.receiveMessages().subscribe(response => {
          this.toastr.success(response.message, response.username + ' sent you message');
          if(this.selectedUser !== null) {
            if(this.selectedUser._id && this.selectedUser._id == response.fromUserId) {
              this.messages.push(response);
              setTimeout( () =>{
                document.querySelector(`.message-thread`).scrollTop = document.querySelector(`.message-thread`).scrollHeight;
              },100);
            }
          }
        });
        /*
        * subscribing for messages ends
        */
      }

      private alignMessage(userId){
        return this.userId ===  userId ? false : true;
      }

      ngOnChanges(changes:any) {

        EmitterService.get(this.selectedUserInfo).subscribe( (selectedUser) => {
            this.selectedUser = selectedUser;
        });

        EmitterService.get(this.conversation).subscribe( (data) => {
          this.messages = data.messages;
        });
      }
    }
```

Explanation:

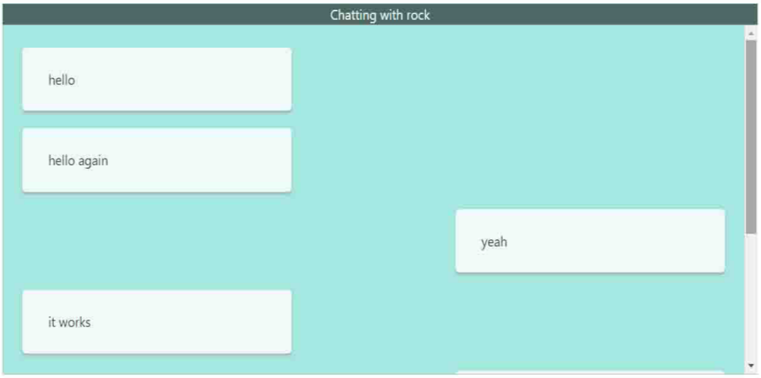
Let's leave the imports and start from bottom, In the last method i.e. ngOnChnages(), I have used the EmitterService in order to get the user information and the messages sent from the home component.

alignMessages() method return true false on the basis of the userid to align the messages as per user. In the listenForMessages() method , I am using the socket service.

In more technical words, Here I am using socket with observables, I am subscribing to the receiveMessages() method which is defined inside the **socketService.ts** file.

So whenever a new message comes from the socket server, the code written inside listenForMessages() method pushes the message into messages array and shows the notification of arrival of new message.

Finally, which looks like below image.



hello

hello again

yeah

it works

Conversation between two users.

SENDING A NEW MESSAGE

To send a new message to other user we will use conversation component itself. Here I will add the textbox and button to send a new message. Open the conversation.component.html and replace the markup from the below markup.

CONVERSATION.COMPONENT.HTML:

```html
HTML
<div class="massege-wrapper">
        <div class="massege-container">

           <div *ngIf="selectedUserName"
              class="opposite-user">
              Chatting with {{selectedUserName}}
           </div>

           <ul class="message-thread">

              <li *ngFor="let message of messages"
                 [class.align-right]="alignMessage(message.toUserId)"
                 >
                    {{ message.message}}
              </li>
           </ul>

        </div>
        <div class="message-typer">
           <textarea
              class="message form-control"
              placeholder="Type and hit Enter"
              [(ngModel)]="message"
              (keyup)="sendMessage($event)"
           ></textarea>
        </div>
```

```
    </div>
```

Now open make above code work, let’s add some functionality behind it. So open the **conversation.component.ts** and add the below method into the conversation class.

CONVERSATION.COMPONENT.TS:

```typescript
export class ConversationComponent implements OnInit {


  ------------
  ------------
  ------------


  private sendMessage(event){
    if(event.keyCode === 13) {
      if(this.message === '' || this.message === null) {
        alert(`Message can't be empty.`);
      }else{

        if (this.message === '') {
          this.toastr.error(`Message can't be empty.`);
        }else if(this.userId === ''){
          this.router.navigate(['/']);
        }else if(this.selectedUser === null){
          this.toastr.error(`Select a user to chat.`);
        }else{

          const data = {
            fromUserId : this.userId,
            message : (this.message).trim(),
            toUserId : this.selectedUser._id,
            toSocketId : this.selectedUser.socketId
          }
          this.messages.push(data);
          setTimeout( () =>{
              document.querySelector(`.message-thread`).scrollTop = document.querySelector(`.message-thread`).scrollHeight;
          },100);

          /*
          * calling method to send the messages
          */
          this.message = null;
          this.socketService.sendMessage(data);
        }
      }
    }
  }


  ------------
  ------------
  ------------

}
```

In the above method, I am using the socket service to send the messages to socket server. First of all, here I am creating an object which comprises of the fromUserID(user id of the logged in user), toUserId(user id of the person chatting t the logged in user), toScoketId and a Message which need to be send.

sendMessage() method defined inside the socket service, which emit the data from client side and at the same I am pushing the message to the messages array.

START A NEW CHAT

To start a new chat, we will provide the list of users, from which logged in user can select any particular user from that list. Here I am using rendering the list users in a pop box (bootstrap Modal popup).

In the HomeComponent I will provide a button to and on click of that button that modal popup will appear.

So let's complete the final markup for HomeComponent.

HOME.COMPONENT.HTML:

```html
<!-- Loading overlay section starts -->
<div bsModal #lgModal="bs-modal" class="modal fade" tabindex="-1" role="dialog" aria-labelledby="myLargeModalLabel" aria-
hidden="true">
  <div class="modal-dialog modal-lg">
    <div class="modal-content">
      <div class="modal-header">
        <h4 class="modal-title pull-left">Start a new Chat</h4>
        <button type="button" class="close pull-right" (click)="lgModal.hide()" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
        <app-new-chat (closeModel)="closeModel($event)"></app-new-chat>
      </div>
    </div>
  </div>
</div>
<!-- Loading overlay section ends -->

<div class="home">
  <!-- header section starts -->
  <div class="header">
    <div class="welcome-user">
      <span>Hello {{username}}</span>
      <div class="home-navigation">
        <ul class="navigation">
          <li (click)="getUsersToChat()" title="Add new chat">
            <i class="fa fa-user-plus" aria-hidden="true"></i>
          </li>
          <li (click)="logout()" title="Logout">
            <i class="fa fa-sign-out" aria-hidden="true"></i>
          </li>
        </ul>
      </div>
    </div>
  </div>
  <!-- header section ends -->

  <div class="chat-body">
    <!-- Messages section starts -->
    <div class="col-md-8">
      <app-conversation [conversation]="conversation" [selectedUserInfo]="selectedUserInfo"></app-conversation>
    </div>
    <!-- Messages section ends -->

    <!-- User chat list section starts -->
    <div class="col-md-4">
      <app-chat-list [conversation]="conversation" [selectedUserInfo]="selectedUserInfo"></app-chat-list>
    </div>
    <!-- User chat list section ends -->
  </div>

</div>
```

Explanation:

As you can see in the above code, I have written the code for modal popup, where I will render the list of users with the help of the **NewChatComponent** component. I will talk NewChatComponent after some time but first let's talk about homeComponent .ts file.

HOME.COMPONENT.TS:

```typescript
TYPESCRIPT
/* Importing from core library starts*/
import { Component, OnInit, Input, OnChanges, ViewChild, ViewContainerRef } from '@angular/core';
import { ActivatedRoute,Router } from '@angular/router';
/* Importing from core library ends*/

import { ChatListComponent } from '../chat-list/chat-list.component';
import { ConversationComponent } from '../conversation/conversation.component';
import { NewChatComponent } from '../new-chat/new-chat.component';

/* Importing ToastsManager library starts*/
import { ToastsManager } from 'ng2-toastr/ng2-toastr';
import { ModalDirective } from 'ng2-bootstrap';
/* Importing ToastsManager library ends*/

/* Importing Application service(i.e. AppService) and http service starts*/
import { AppService } from './../services/app.service';
import { HttpService } from './../services/http.service';
import { SocketService } from './../services/socket.service';
import { EmitterService } from './../services/emitter.service';
/* Importing Application service(i.e. AppService) and http service ends*/

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css'],
})
export class HomeComponent implements OnInit {

  /*
  * Chat and message related variables starts
  */
  private userId = null;
  private username = null;
  private userSocketId =null;
  /*
  * Chat and message related variables ends
  */

  @ViewChild(ChatListComponent) chatListComponent: ChatListComponent
  @ViewChild(ConversationComponent) conversationComponent: ConversationComponent
  @ViewChild(NewChatComponent) NewChatComponent: NewChatComponent
  @ViewChild('lgModal') public lgModal: ModalDirective;

  private conversation = 'CONVERSATION';

  constructor(
    private socketService : SocketService,
    private httpService : HttpService,
    private appService : AppService,
    private route :ActivatedRoute,
    private router :Router,
    private _emitterService: EmitterService
  ) { }

  ngOnInit() {
    /*
    * getting userID from URL using 'route.snapshot'
    */
    this.userId = this.route.snapshot.params['userid'];
    if(this.userId === '' || typeof this.userId == 'undefined') {
      this.router.navigate(['/']);
    }else{
      /*
```

```
        * Checking if user is logged in or not, starts
        */
        this.appService.userSessionCheck(this.userId,( response )=>{
          this.username = response.username;
          /*
          * making socket connection by passing UserId.
          */
          this.socketService.connectSocket(this.userId);

          this.chatListComponent.getChatList();

          this.conversationComponent.listenForMessages();
        });
        /*
        * Checking if user is logged in or not, starts
        */
      }
    }

    private getUsersToChat(){
      this.NewChatComponent.getUsersToChat();;
      this.lgModal.show();
    }

    private closeModel(event){
      this.lgModal.hide();
      this.chatListComponent.getChatList();
    }

    private logout(){
      this.socketService.logout({userId : this.userId}).subscribe(response => {
        this.router.navigate(['/']); /* Home page redirection */
      });
    }
  }
```

Explanation:

The method getUsersToChat() is defined inside the NewChatComponent, where I will fetch list user with whom we can start a new chat. And as the name suggests, the closeModel() will close the modal popup.

Only for our angular application only one component left to understand and i.e. NewChatComponent component. In this component I will render the list of the users and we will write method to start a new chat with new users. Let's look into the NewChatComponent component.

Let's start with the new-chat.component.html file, what we have there.

NEW-CHAT.COMPONENT.HTML:

```
                                                                    HTML
<div class="user-list-wrapper">
  <ul *ngIf="usersToChat.length !== 0" class="new-user-list">

    <li *ngFor="let user of usersToChat"
      (click)="selectToChat(user)">
      <div class="new-chat-username">
        {{user.username}}
        <span *ngIf="user.online=='Y'" class="online-user" title="Online"></span>
        <span *ngIf="user.online=='N'" class="offline-user" title="Offline"></span>
      </div>
      <div class="new-chat-textbox" [class.new-chat-textbox-seleted]="selectedUserToChat(user._id)">
        <br>
        <textarea class="form-control" [(ngModel)]="message" placeholder="Write something here"></textarea>
```

```html
            <br>
            <button class="btn btn-primary" (click)="sendNewMessage()">Send</button>
          </div>

        </li>
      </ul>

      <div *ngIf="usersToChat.length === 0"
        class="alert alert-info text-center">
        <strong> No new user to chat. </strong>
      </div>
    </div>
```

Explanation:

In the above markup, we are looping users available to chat and when you select a user to chat I will show the message box where a user a can and type a new message and send a message.

Well that sweet and short html and now let see what we have in our typescript code. Open the **new-chat.component.ts** and write down below code.

NEW-CHAT.COMPONENT.TS:

```typescript
TYPESCRIPT
/* Importing from core library starts*/
import { Component, OnInit, Output, EventEmitter, ViewChild, AfterViewInit, ViewContainerRef } from '@angular/core';
import { ActivatedRoute,Router } from '@angular/router';
/* Importing from core library ends*/

/* Importing ToastsManager library starts*/
import { ToastsManager } from 'ng2-toastr/ng2-toastr';
/* Importing ToastsManager library ends*/

/* Importing Application service(i.e. AppService) and http service starts*/
import { AppService } from './../services/app.service';
import { HttpService } from './../services/http.service';
import { SocketService } from './../services/socket.service';
/* Importing Application service(i.e. AppService) and http service ends*/


@Component({
  selector: 'app-new-chat',
  templateUrl: './new-chat.component.html',
  styleUrls: ['./new-chat.component.css'],
})
export class NewChatComponent {

  private userId = null;
  private selectedUserId = null;
  private selectedSocketId = null;
  private message = null;
  private usersToChat = [];

  @Output('closeModel') closeModel = new EventEmitter<boolean>();


  constructor(
    private httpService : HttpService,
    private appService : AppService,
    private socketService : SocketService,
    private route :ActivatedRoute,
    private router :Router,
    private toastr: ToastsManager,
    private _vcr: ViewContainerRef
  ) { }

  public getUsersToChat() {
    this.userId = this.route.snapshot.params['userid'];
    if(this.userId === '' || typeof this.userId == 'undefined') {
      this.router.navigate(['/']);
    }else{
      this.appService.getUsersToChat({
        userId : this.userId
      },(error,response)=>{
        if(error) {
          this.toastr.info('No one to chat, try after some time.');
        }else{
          this.usersToChat = response.chatList;
        }
      });
    }
  }

  private selectToChat(user):void{
    this.selectedUserId = user._id;
    this.selectedSocketId = user.socketId;
  }
  /*
  * Method required for UI to indicate the selected user starts.
  */
  private selectedUserToChat(userId:string):boolean{
    if(!this.selectedUserId) {
      return false;
    }
    return this.selectedUserId ===  userId ? true : false;
  }

  private sendNewMessage(){
    if(this.message === '' || this.message === null) {
      this.toastr.error('Message cant be empty.');
    }else{

      if (this.message === '') {
        this.toastr.error('Message cant be empty.');
      }else if(this.userId === ''){
        this.router.navigate(['/']);
      }else if(this.selectedUserId === null){
        this.toastr.error('Select a user to chat.');
      }else{

        const data = {
          fromUserId : this.userId,
          message : (this.message).trim(),
          toUserId : this.selectedUserId,
          toSocketId : this.selectedSocketId,
          startNewChat : true
        }

        /*
        * calling method to send the messages
        */
        this.message = null;
        this.socketService.sendMessage(data);
        this.closeModel.emit(true);
      }
    }
  }
}
```

Explanation:

The **NewChatComponent** class I have defined four methods which are explained in below in depth.

getUsersToChat(): This method fetches the list of user with whom logged in user start a new chat with the appService class.

selectToChat(): This method is used to highlight the selected row.

selectedUserToChat(): This method will display message box in which a user can write a message and button to send the message.

sendNewMessage(): In this method, as the name suggest we will send a message in order to start a new chat. Here I am using sendMessage() method which defined inside the **socketService** class. And once the message is sent then I will close the modal popup.

CHAPTER 12. THE END

Now we have completed the development of the server as well as Angular application. We have implemented the some of the chatting application's important features.

But it's not the end yet, I will constantly update this e-book. Also, I will add more features in this chat application. You can always suggest me any new feature, which you would like to see in this chat application. You can contact me from here.

ABOUT THE AUTHOR

My name is Shashank Tiwari, Software engineer by profession from Mumbai, India.

www.codershood.info is a platform where I share my experiences and knowledge through work or the information learned during my past time. I am here to exchange substances with the help of the posted articles and your suggestions are welcome to bring on enhancements or queries related to an article.

Blogging and exploring new things is my passion which motivates me to write and share, how to build a complex application, shot tips and tutorials on new languages and framework.

I also listen to requests for tutorials and suggestions here.