Joao Pereira
19354106

Demo Link - https://youtu.be/CUvnn_88npE

Reference Disclaimer - The program was implemented with the aid of https://www.youtube.com/watch?v=UymGJnv-WsE Video was followed through for an understanding of the topic as it wasn't of my knowledge. Research was also conducted and through countless blogs, articles and videos, this reference was the one that got my understanding to the fullest.

Sockets

1. What are the three types of network sockets and what are their differences?
2. What is the process of creating a network socket?
3. The desktop computer with the IP address 192.168.0.5 is running multiple applications

   What are the steps involved when the user opens their browser and visits the website discord.com, make sure to take account of the protocols involved in each step?

Websockets

1. What are the differences between a websocket and a normal network socket?
2. Websockets provide full-duplex communication, what does this mean, and how did we achieve similar functionality before WebSockets were released.
3. Detail the steps, requests and protocols involved when you:
   - Start your app server
   - Visit it in a browser
   - Send a message

Sockets

1

Three types of network sockets are Stream (SOCK_STREAM), Datagram (SOCK_DGRAM) and Raw (SOCK_RAW) sockets [3].

Stream sockets are utilized through the use of TCP data to communicate back and forth between a client and a server. Their socket type is SOCK_STEAM [3]. Since they're above TCP, they are able to operate/communicate across any network in a bidirectional, reliable and sequenced manner of data [1]. If an error occurs when delivering data, the user/sender receives an error message [2]. Once a connection has been accomplished, the data can be read and written through the form of byte streams to the socket [1]

Datagram uses UDP data to communicate [3]. Their socket type is SOCK_DGRAM [3]. A Datagram socket consists of a bidirectional flow of data/messages [1]. Datagram sockets have the ability to receive messages in duplication and in an order that is different to the sending sequence [1]. *Unlike* Stream sockets, they do not need any open connections, to construct a socket consisting of the destination info and to transfer the data through the use of the datagram protocol [2].

Raw sockets have socket type of SOCK_RAW [3]. Raw sockets are often datagram oriented [1]. Their main purpose is to aid developers in creating new communication protocols or those who have access to the crypto facilities of existing protocols [2]. They are not generally for the general public/user. There are two types of raw sockets [5].

1. The first raw socket type utilizes a known protocol type (such as ICMP) that is written within the header recognized by a Winsock provider
2. The second raw socket type allows any protocol type to be defined. An example of this would be SCTP (Stream Control Transmission Protocol).

2

We start off the process by implementing a server that requests the operating system to create a socket. It then requests to bind to a port and an address and then to check for possible connections.

It then turns to the client-side where the client requests the operating machine to create a socket and then to form a connection to the remote host and port.

The server then accepts the connection and assigns a new socket to the local port whilst also setting the remote endpoint to the port and address of the client. For every single connection formed to the same port, the server has a designated socket that reads and writes to communicate with the client who is connected. However, the server needs a new socket to ensure that it continuously listens to the original socket for requests [6] whilst also staying connected to the client.

Once the connection is accepted by the server, the client will be able to communicate back and forth with the server since a socket will be created upon acceptance.

3

The computer with IP address 192.168.0.5 is running on multiple applications. First of all, the mentioned IP address is the routers address in which the computer is connected to the network and which will be used to send data across.

Now, what are the steps involved with a user opening a browser and searching for Discord.com?

1. The browser extracts the Discord URL and domain name.

2. The browser checks the DNS entry to find the corresponding IP address of Discord.
3. It searches for a cache within a list of cache, it searches through the following and performs a DNS lookup [12]:
    - Browser Cache
    - Operating Systems Cache
    - Router Cache
    - ISP Cache
4. If a cache is not found, ISP (Internet Service Provider) has a DNS server which begins a DNS query to search for the IP address of the server that hosts the domain name. Each request that is sent is consisted of small data packages that provide request information of the content.
5. IP address of the DNS server is found using ARP (Address Resolution Protocol) [13].
6. Once the IP address of the server is obtained, port 80/443 is requested and a TCP connection is requested.
7. The request then travels to the network layer where it be will be packing the TCP header. It also travels to the transport layer and data link layer to fill the IP header and ethernet frame header respectfully [13].
8. Browser and server form a TCP connection by using a 3-way handshake or SYN (synchronize) and ACK (acknowledge) messages.
9. The transport layer then secures the 3-way handshake.
10. HTTP request is then sent by the browser to Discord's server. It is either performed by a GET or POST request.
11. The host's operating system then receives the request and responds in the format of either JSON, HTML or XML.
12. Discord receives the request and a HTTP response is then sent out by the server as well as the status (such as 200 OK).
13. The browser then beings to parse the response to display it. HTML bodies that include CSS and JavaScript will perform additional calls back to the server.
14. The browser then displays and renders Discord's HTML content to the user.
15. Discord then determines who the user is and what information they need to send back.

Websockets

1

Normal network sockets typically run over TCP/IP protocols however they are not restricted by HTTP protocols like Websockets. They are used to receive and send data between the client and the server.

On the other hand, Websockets are run by a server connecting through protocols similar to HTTP that are built on TCP/IP. They are utilized just as normal network sockets are, to send and receive data, however, they're more or less integrated for continuous real-time messages. Therefore, they're much quicker, and each connection is not created from the start and destroyed after use [7].

2

Full-Duplex Communication is the transmission of data being transmitted through both directions or two ways at the same time on one single carrier. An example of this would be a phone call, where there is a receiver and a sender, and in this case where both transmit data over to another.

Websockets integrate full-duplex-communication as stated above, they can send and receive data between a client and a server in a persistent, continuous and real-time manner.

Before Websockets was first introduced in 2009, we reached a similar functionality through HTTP Polling, Long-Polling, AJAX and through the BOSH protocol [9], [10].

HTTP Polling was utilized as a solution to Cross Frame Communication where the content would be loaded into the current page and the user would be under the impression of being in the application-style experience [10]. It operated by loading new pages under hidden frames. However, one big issue that came with CFC were made up of requests for the server and therefore it had faults with providing new and consistent information to the user. The solution was then introduced as HTTP Polling. HTTP Polling served to poll the server at often intervals for the client [8]. For each request that was formed, new connections were created.

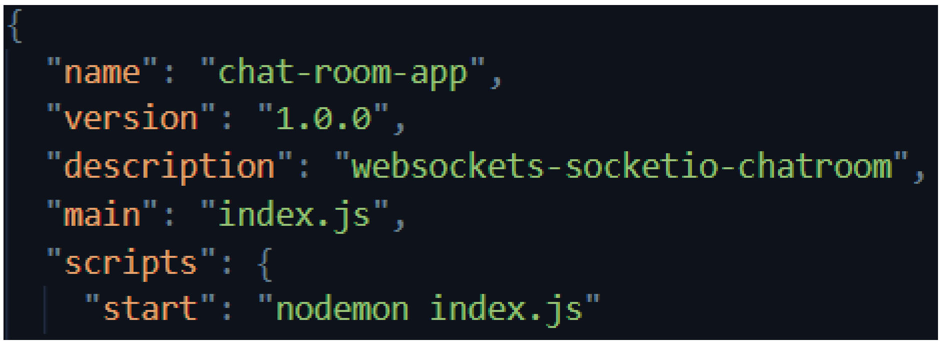
Long Polling was soon after introduced as an advanced and improved method of HTTP Polling. The client would form a connection with the server and the server would, in turn, hold the connection open until it was closed. When new, additional information was gathered for the client, it would send the data through and close the connection. The client would then once again re-form a connection and wait for any upcoming data. The biggest issue with this would be that when the user would be in the re-connecting phase, data could be unreliable and out of data.

AJAX (Asynchronous JavaScript and XML) was a commonly used method before Websockets were brought into the world. A connection would be formed through JavaScript only when the data would be required and then the server would respond with that requested data [10]. There would be no need for Cross Frame Communication. Users' were able to view a website while not needing to wait for responses, this was referred to as non-blocking. The only issue with AJAX is the fact that it performs poorly when dealing with large amounts of data, it creates a bottleneck.

.Bidirectional Streams Over Synchronous HTTP or BOSH was a protocol to transport two entities through the use of synchronous HTTP request/response pairs without the use of polling [11]. BOSH was more advanced/enhanced than other techniques such as AJAX as it avoided HTTP polling by avoiding sorting chunked HTTP responses, known as Comet. BOSH is operated by the client submitting a HTTP request and the server would then postpone a response until it ad data to send. Once a response was formed, the client then makes another request on the same connection so the server has access to send data to the client without ever having to poll. Also if a client needed to send data to the server while waiting for a response from the server, it would open a second HTTP connection.

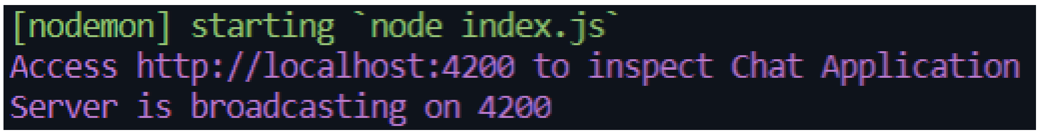
3

To start off the application, open up your terminal to the directory the files are located and proceed to run the program by the commands "yarn start" or "sudo npm run start". The scripts to run the application are located within the package.json file.



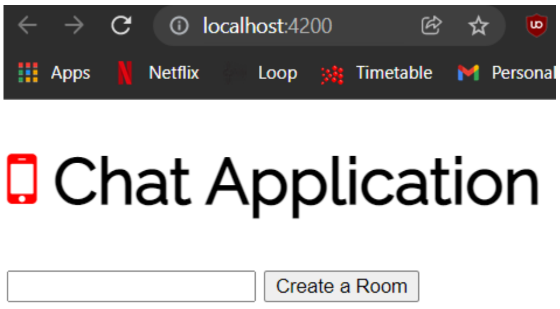
```
{
  "name": "chat-room-app",
  "version": "1.0.0",
  "description": "websockets-socketio-chatroom",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js"
```

Once the commands are executed and the program is executing successfully, you should be presented with a message applying that the server is being broadcasted and what port it is being broadcasted on with its link as well.



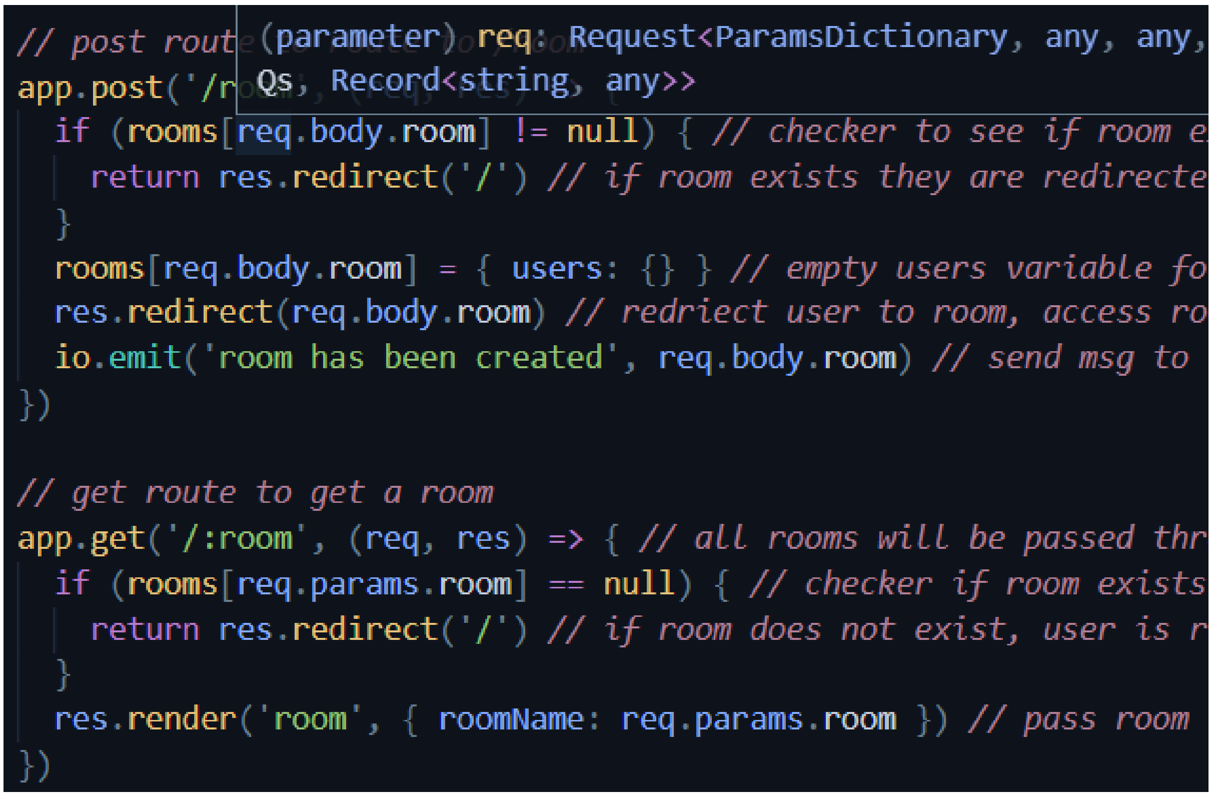
```
[nodemon] starting `node index.js`
Access http://localhost:4200 to inspect Chat Application
Server is broadcasting on 4200
```

Follow the link or simply search for the port on your browser and the program and it's HTML and JavaScript content will be displayed. As such:



Once on the home page, you have the ability to “Create a Room”. Simply type the name of the room you desire to create and then click the “Create a Room” button.

The room implementation is a POST request.

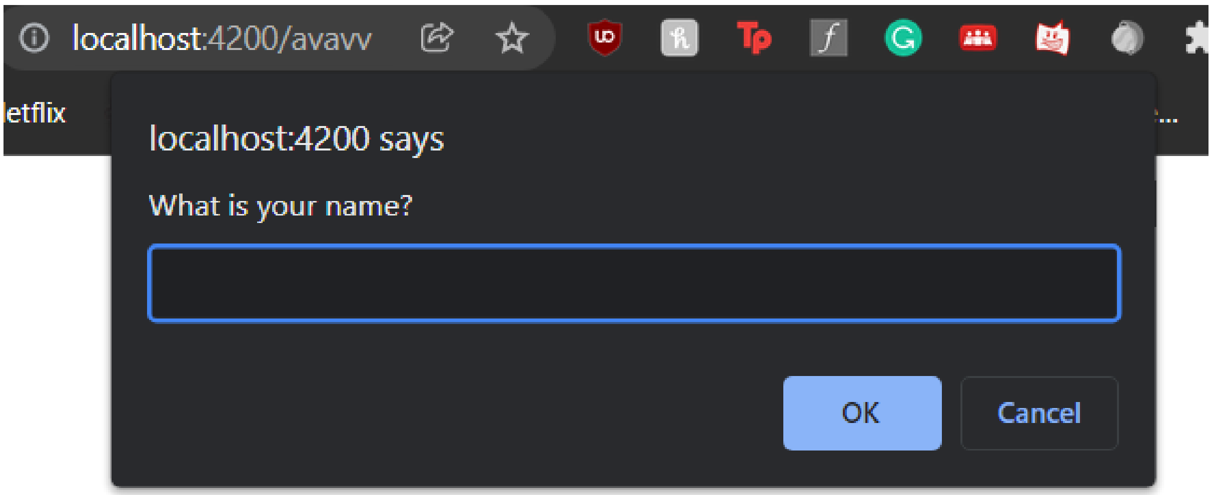


```
// post rout(parameter) req: Request<ParamsDictionary, any, any,
app.post('/r Qs, Record<string, any>>
  if (rooms[req.body.room] != null) { // checker to see if room e
    return res.redirect('/') // if room exists they are redirecte
  }
  rooms[req.body.room] = { users: {} } // empty users variable fo
  res.redirect(req.body.room) // redriect user to room, access ro
  io.emit('room has been created', req.body.room) // send msg to
})

// get route to get a room
app.get('/:room', (req, res) => { // all rooms will be passed thr
  if (rooms[req.params.room] == null) { // checker if room exists
    return res.redirect('/') // if room does not exist, user is r
  }
  res.render('room', { roomName: req.params.room }) // pass room
})
```

When you create a room the program implements a POST request to create the room and then sends all the information/data (“room has been created”) by emitting that information to the user.

If a use joins a room through the home page or URL, the program performs a GET method to the server to search for the room and see if it exists and if it doesn’t then the user will be redirected to the home page. Once a user joins a room, they will be presented with a pop-up requesting a user name as an identifier.

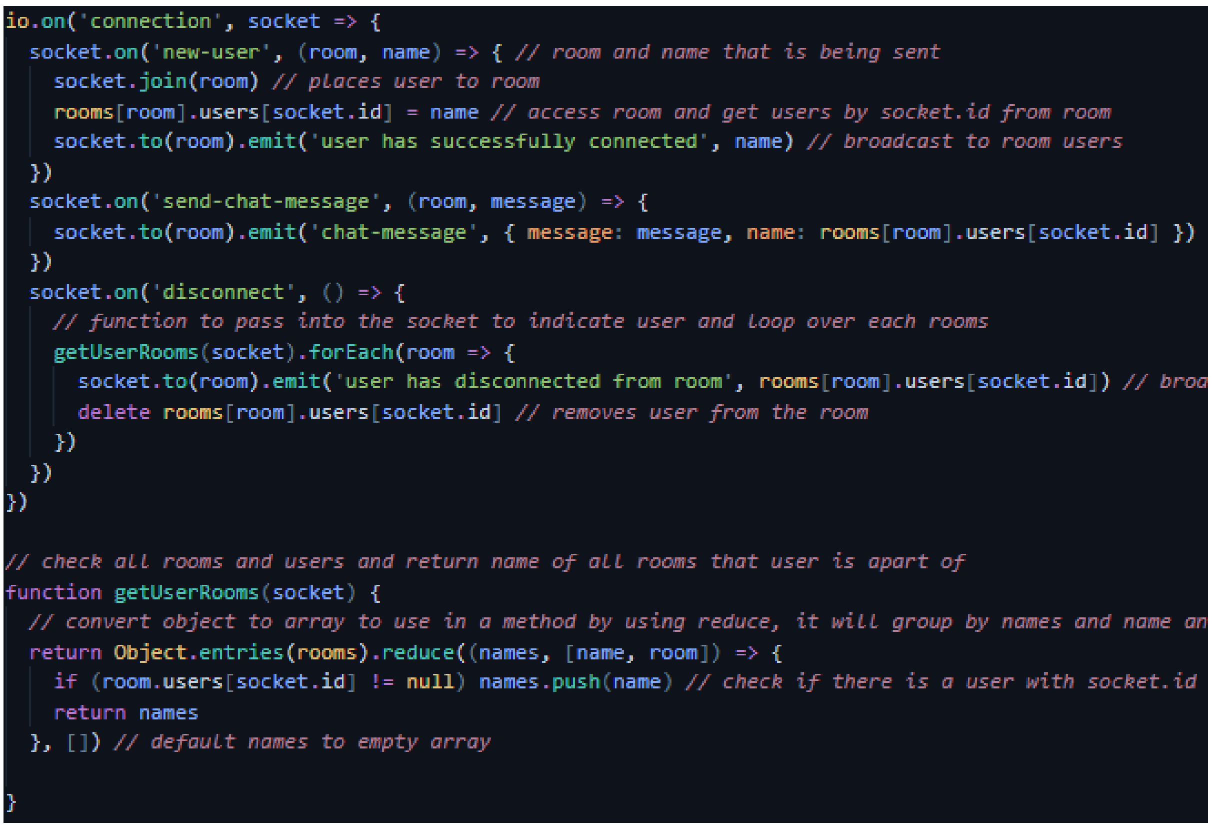


localhost:4200/avavv

localhost:4200 says

What is your name?

Once in the room and the user name is created you are able to send receive messages between other clients who also decide to join the room. This is performed through the use of the function:



```
io.on('connection', socket => {
  socket.on('new-user', (room, name) => { // room and name that is being sent
    socket.join(room) // places user to room
    rooms[room].users[socket.id] = name // access room and get users by socket.id from room
    socket.to(room).emit('user has successfully connected', name) // broadcast to room users
  })
  socket.on('send-chat-message', (room, message) => {
    socket.to(room).emit('chat-message', { message: message, name: rooms[room].users[socket.id] })
  })
  socket.on('disconnect', () => {
    // function to pass into the socket to indicate user and loop over each rooms
    getUserRooms(socket).forEach(room => {
      socket.to(room).emit('user has disconnected from room', rooms[room].users[socket.id]) // broa
      delete rooms[room].users[socket.id] // removes user from the room
    })
  })
})

// check all rooms and users and return name of all rooms that user is apart of
function getUserRooms(socket) {
  // convert object to array to use in a method by using reduce, it will group by names and name an
  return Object.entries(rooms).reduce((names, [name, room]) => {
    if (room.users[socket.id] != null) names.push(name) // check if there is a user with socket.id
    return names
  }, []) // default names to empty array

}
```

As you can see through the use of the two functions, they are performing diffeent actions such as a user joining, a user disconnecting and the chat messages being communicated between clients and server in a partcular room.

References

1. https://docs.oracle.com/cd/E19455-01/806-1017/sockets-4/index.html
2. https://www.educba.com/types-of-socket/
3. https://loop.dcu.ie/mod/resource/view.php?id=1736205&redirect=1
4. https://www.careerride.com/Networking-what-is-stream-socket.aspx
5. https://docs.microsoft.com/en-us/windows/win32/winsock/tcp-ip-raw-sockets-2
6. https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html
7. https://stackoverflow.com/questions/4973622/difference-between-socket-and-websocket
8. https://blog.pusher.com/what-came-before-websockets/
9. https://stackoverflow.com/questions/51726431/how-was-real-time-browser-chat-implemented-before-websockets
10. https://dzone.com/articles/pre-websocket-workarounds
11. https://en.wikipedia.org/wiki/BOSH_(protocol)

12. https://www.geeksforgeeks.org/what-happens-when-we-type-a-url/
13. https://www.quora.com/What-are-the-series-of-steps-that-happen-when-a-URL-is-requested-from-the-address-field-of-a-browser