# *Express Overview*

# I.BACKEND OVERVIEW

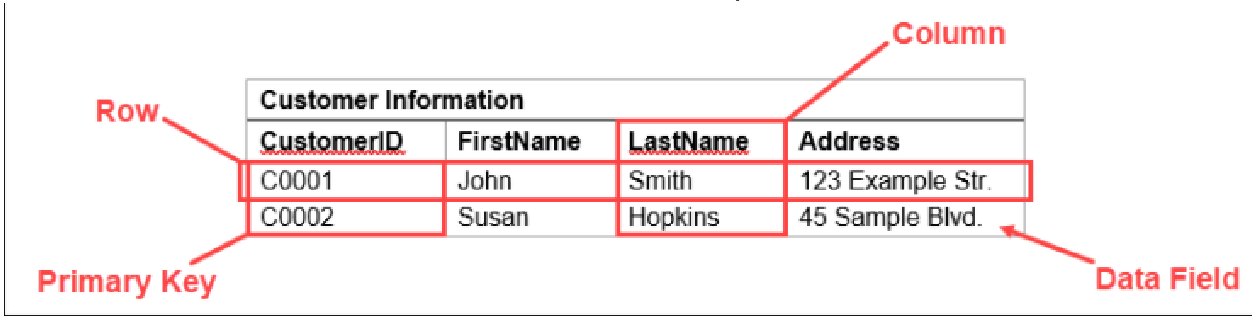
### What is Backend?

Any Parts of the Website that the user can't see, have a lot of logic written and interact with a database to give the required data to the frontend
The backend of the Website consists of Server Application Database
**Server:** A server is nothing but a machine connected to a network, and having an application running on it
**Application :** Application does the task of receiving requests from Frontend and giving back responses to Frontend
**DataBase:** Organized Collection of Data So that data can be easily accessed and managed
Example : MySqL, DynamoDB,MongoDB
**Some backend languages are: Python, PHP, Java, Ruby**

### What are Various types of Databases?

#### A.Relational Database/SQL Database

1 Relational database stores data in rows and columns in form of Table
2 It is used to handle data coming in low-velocity,
3 It supports Complex Transactions
4 They are compliant with ACID (Atomicity, Consistency, Isolation, Durability) Properties
5 Data arrives from one and a few locations
**Example: MySQL, PostgreSQL,MariaDB,SQLite**
How Data is Stored in a Relational Database System?

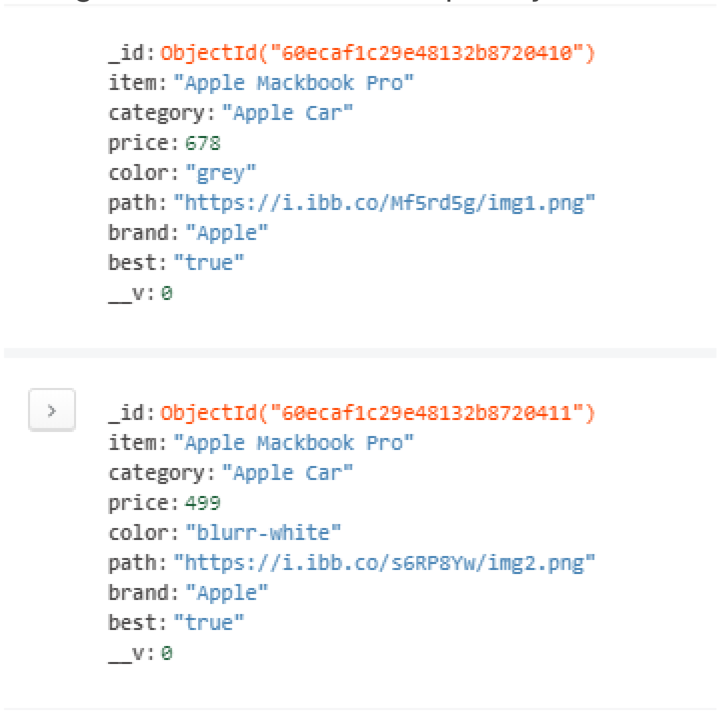

Column
Row
Primary Key
Data Field

| Customer Information | | | |
|---|---|---|---|
| CustomerID | FirstName | LastName | Address |
| C0001 | John | Smith | 123 Example Str. |
| C0002 | Susan | Hopkins | 45 Sample Blvd. |

Data Stored in MYSQL

#### B Non-Relational Database/NoSQL Database

1 NoSQL database are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph.
2 It is used to handle data coming in high-velocity.
3 It supports Simple Transactions.
4 NoSQL databases deemphasize the principles of ACID (atomicity, consistency, isolation, and durability).
5 Data arrives from many Locations.
**Example : MongoDB,Dynamodb**
How Data is stored in Non-relation Database System?
In MongoDB, data is stored as documents. These documents are stored in MongoDB in JSON (JavaScript Object Notation) format.



```
_id: ObjectId("60ecaf1c29e48132b8720410")
item: "Apple Mackbook Pro"
category: "Apple Car"
price: 678
color: "grey"
path: "https://i.ibb.co/Mf5rd5g/img1.png"
brand: "Apple"
best: "true"
__v: 0
```

```
_id: ObjectId("60ecaf1c29e48132b8720411")
item: "Apple Mackbook Pro"
category: "Apple Car"
price: 499
color: "blurr-white"
path: "https://i.ibb.co/s6RP8Yw/img2.png"
brand: "Apple"
best: "true"
__v: 0
```

Data Stored in MongoDb

## II. INTRODUCTION TO NODEJS

### What is Nodejs?

Node.js is an open-source, cross-platform, Javascript runtime environment built on Chrome's V8 Javascript engine that executes Javascript Code outside a web browser.
**Open source:** Open source mean code that is designed to be publicly accessible
**Cross-Platform:** We can run nodejs code on various platforms like Mac, Windows, or Where javascript is installed, The code we wrote in one machine can run on another machine.
**Run-Time Environment:** The runtime environment is just the environment in which your application is running.
**Nodejs is asynchronous and single-threaded in nature**

## What is a Package?

**Package:** A package in Node. js contains all the files you need for a module.
**Modules:** are JavaScript libraries that you can include in your project.

## What is npm in Node.js?

NPM stands for **Node Package Manager**. It provides the following two main functionalities.

1. It works as an Online repository for node.js packages/modules Where everyone can share Tools written in Javascript
2. It works as a Command-line utility to install packages, do version management and dependency management of Node.js packages. NPM comes bundled along with Node.js installable.

Command used to check npm version
```
npm --version
```
NPM helps to install any Node.js module using the following command.
```
npm install <Module Name>
```
Example:
```
npm install express (express node.js framwork)
```

## What is the event loop in node js?

The event loop is what allows Node. js to perform non-blocking I/O operations — despite the fact that JavaScript is single-threaded

## When should we use nodejs and when not?

### When to use Node.js

It is ideal to use Node.js for developing streaming or event-based real-time applications that require less CPU usage such as

1. Game servers :
2. Chat Applications
3. Advertisement and Streaming services
4. Good for a collaborative Environment: The “Event loop” feature of Node.js enables it to handle multiple events simultaneously without getting blocked. when people work together

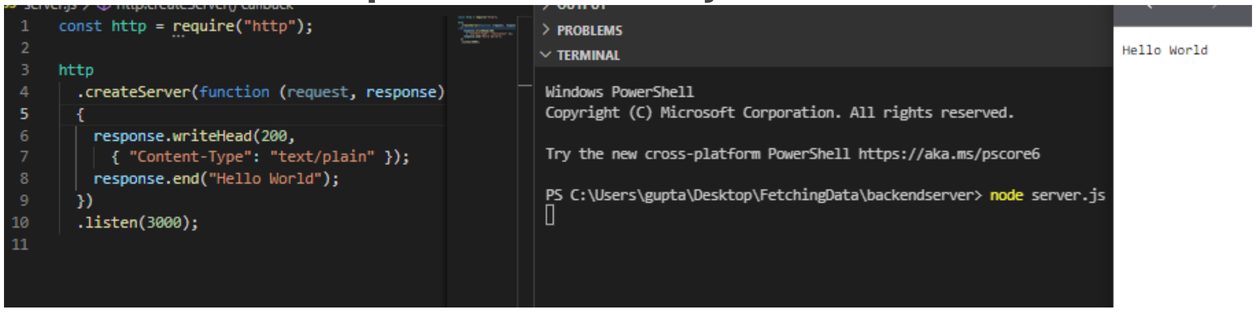
### When not to use Node js

Node js is a single-threaded, so we should not use it for cases where the application requires a long processing time. If the server is doing some calculations, it won’t be able to process any other requests. Hence, Node.js is best when processing needs less CPU time.

## What are the key features of Node.js?

**Asynchronous event-driven** All APIs of Node.js are asynchronous. Thus new requests will not wait for the response from the previous requests.
**Fast in Code execution** Node.js uses the V8 JavaScript Runtime engine, Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence process requests with node.js is much faster
**Active community For Node.js** The active community always keeps the framework updated with the latest trends in web development.
**No Buffering** Node.js applications never buffer any data. They simply output the data in chunks.

## How to create a simple server in Node.js?



```javascript
const http = require("http");

http
  .createServer(function (request, response)
  {
    response.writeHead(200,
      { "Content-Type": "text/plain" });
    response.end("Hello World");
  })
  .listen(3000);
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\gupta\Desktop\FetchingData\backendserver> node server.js
```

```
Hello World
```

# III EXPRESS OVERVIEW

## What is a framework?

A framework works as a kind of support structure for something to be built on top of.

## What is Express?

Express is a framework built on top of Node.

## What are the advantages of using Express?

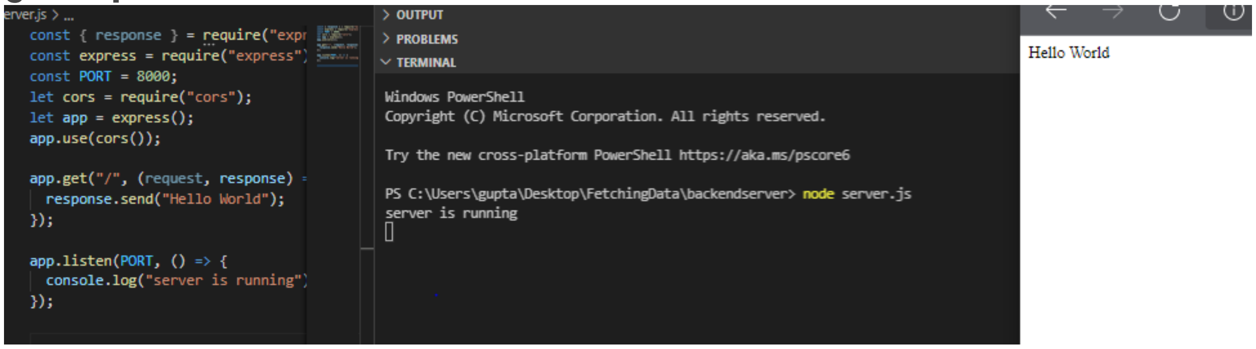
1. With a myriad of HTTP utility methods and middleware , creating a robust API is quick and easy .
2. Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know .
3. Many popular frameworks are based on Express .
4. Express helps you manage everything, from routes to handling requests
5. **Open Source Community** : It has an open-source community,so the code is always reviewed and improved .
6. Express Framework is reliable and has a huge community around .

## How to install and write code in Express?

## Installation

```
npm i express
```

### get request

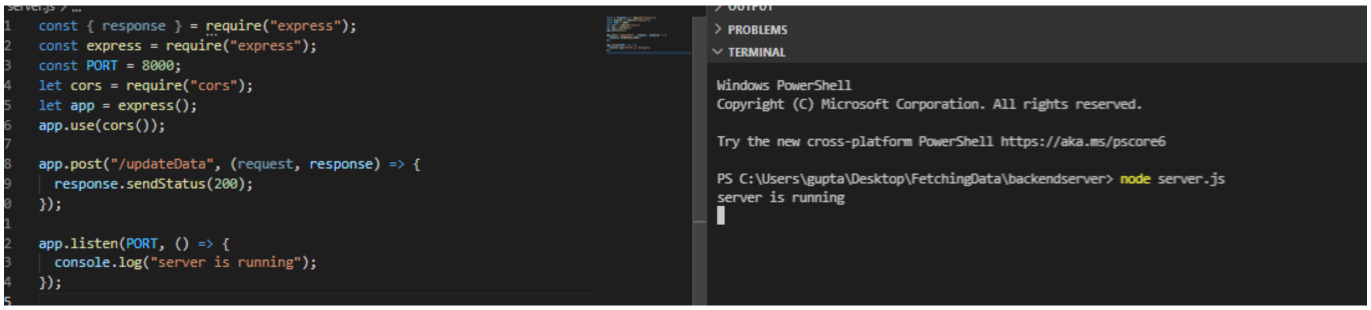


```
const { response } = require("expr
const express = require("express")
const PORT = 8000;
let cors = require("cors");
let app = express();
app.use(cors());

app.get("/", (request, response) =
  response.send("Hello World");
});

app.listen(PORT, () => {
  console.log("server is running")
});
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\gupta\Desktop\FetchingData\backendserver> node server.js
server is running
```

```
Hello World
```

### post request



```
const { response } = require("express");
const express = require("express");
const PORT = 8000;
let cors = require("cors");
let app = express();
app.use(cors());

app.post("/updateData", (request, response) => {
  response.sendStatus(200);
});

app.listen(PORT, () => {
  console.log("server is running");
});
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\gupta\Desktop\FetchingData\backendserver> node server.js
server is running
```

# IV USING MODULES

### What are Modules?

small units of independent, reusable code that is desired to be used as the building blocks.

### What is exports in node.js?

When we want to export multiple variables/functions from one module to another, we use exports.

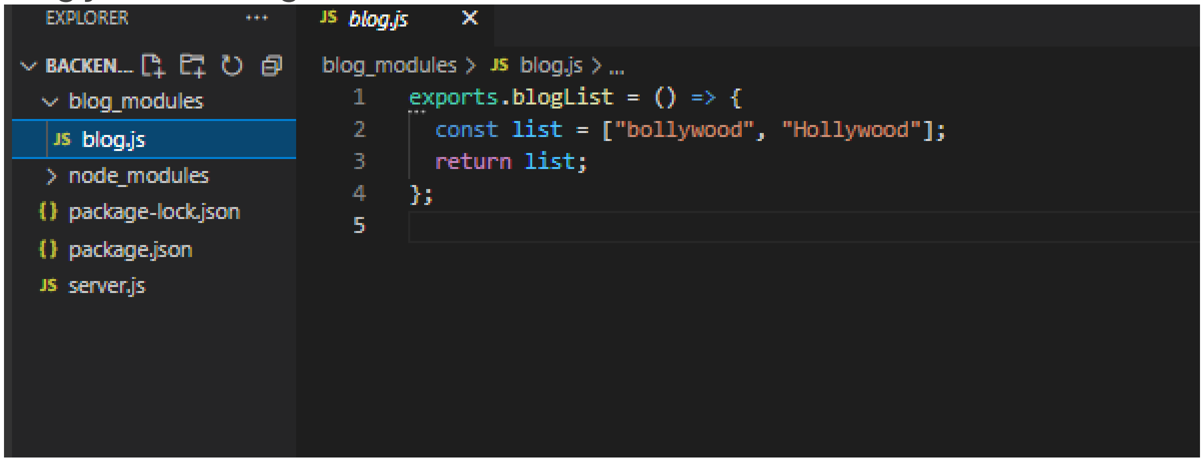
### What is require in node.js?

In NodeJS, require() is a built-in function to include external modules that exist in separate files. require() statement basically reads a JavaScript file, executes it, and then proceeds to return the export object.

### How to export modules from separate .js files and import them in server.js?

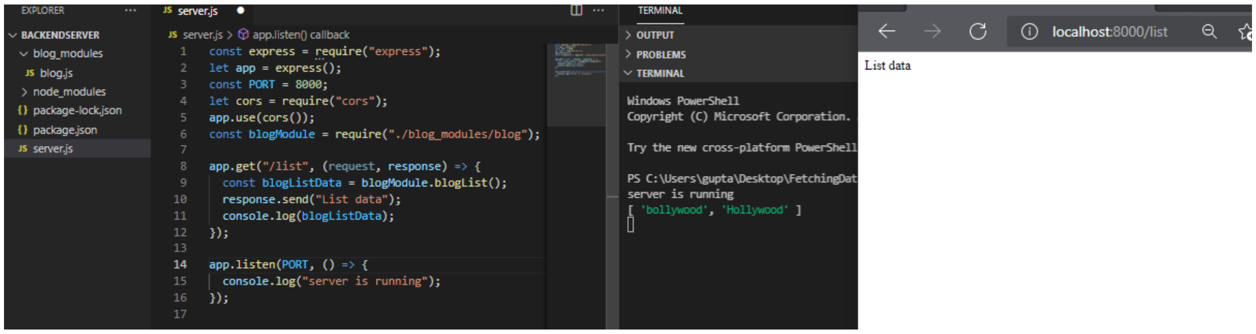
**Code For Exporting Modules from separate.js Files and importing them in server.js**

**Blog.js inside blog_modules**



```
exports.blogList = () => {
  const list = ["bollywood", "Hollywood"];
  return list;
};
```

**Server.js and output console**



```
const express = require("express");
let app = express();
const PORT = 8000;
let cors = require("cors");
app.use(cors());
const blogModule = require("./blog_modules/blog");

app.get("/list", (request, response) => {
  const blogListData = blogModule.blogList();
  response.send("List data");
  console.log(blogListData);
});

app.listen(PORT, () => {
  console.log("server is running");
});
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation.

Try the new cross-platform PowerShell

PS C:\Users\gupta\Desktop\FetchingDat
server is running
[ 'bollywood', 'Hollywood' ]
```

List data

## V.SERVER ROUTING

**Route is url path or pattern**

### What is Routing and how it works ?

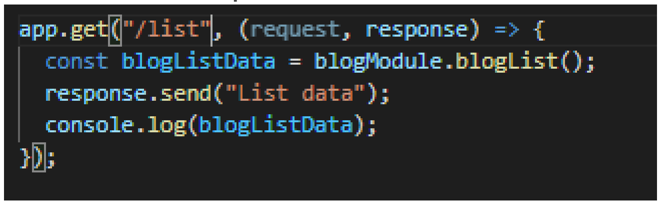
Routing refers how an application responds to a client request to a particular route, URL, or path and a specific HTTP request method (GET, POST, etc.).
We define routing using methods of the Express object (app) that correspond to HTTP methods

**For example :**

**app.get()** to handle GET requests
**app.post()** to handle POST requests
**app.all()** to handle all HTTP methods
**app.use()** to specify middleware

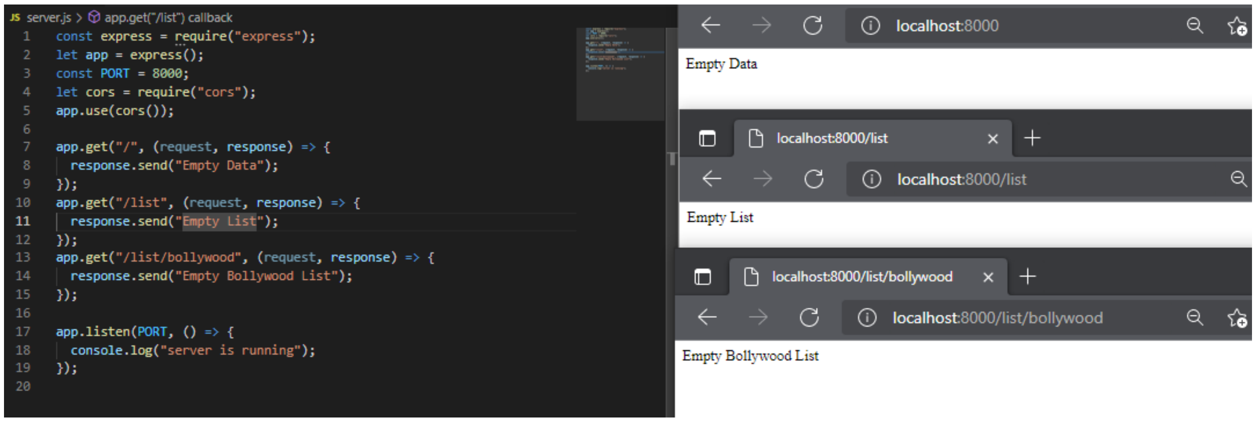
These routing methods listens for requests that match the specified routes, and when it detects a match, it calls the specified callback function In these way routing works .

**Example** :when app.get() methods listen for request that match with the route (/list) it calls a callback function and then send the response to the frontend for a particular route



```
app.get("/list", (request, response) => {
  const blogListData = blogModule.blogList();
  response.send("List data");
  console.log(blogListData);
});
```

**Routing is defined in this way**


server.js > app.get("/list") callback

```
const express = require("express");
let app = express();
const PORT = 8000;
let cors = require("cors");
app.use(cors());

app.get("/", (request, response) => {
  response.send("Empty Data");
});
app.get("/list", (request, response) => {
  response.send("Empty List");
});
app.get("/list/bollywood", (request, response) => {
  response.send("Empty Bollywood List");
});

app.listen(PORT, () => {
  console.log("server is running");
});
```

localhost:8000

```
Empty Data
```

localhost:8000/list

```
Empty List
```

localhost:8000/list/bollywood

```
Empty Bollywood List
```

# ***MiddleWare &Authentication***

## I.MIDDLEWARE OVERVIEW

### What is Middleware and why developers need middleware?

Middleware is a function that accesses the request object (req), response object (res), and access next middleware function in application's request-response cycle

Or

It's nothing but a function that runs even before the call goes to the API for which it is meant to be .

**Middleware is needed because**

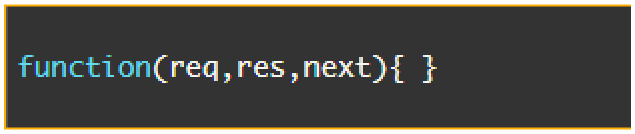
1. Middleware helps developers build applications more efficiently
2. It acts as the connective tissue between applications, data, and users.

3. Middleware functions can perform the following tasks:

- Make changes to the request and the response objects.
- Execute any code.
- End the request-response cycle.
- Call the next middleware in the stack.

**Middleware Function**

All middleware functions in Express.js accept three arguments following the Request, response, and next.

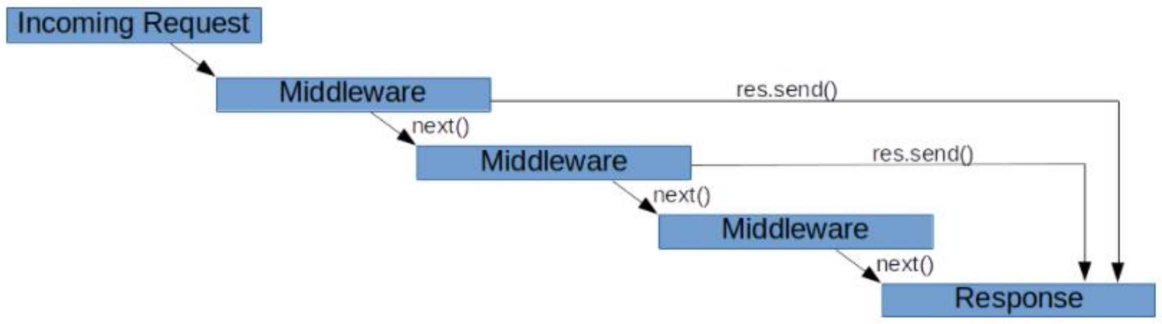


```
function(req,res,next){ }
```

The first argument,req, is shorthand for the request object with built-in properties to access data from the client-side and facilitate HTTP requests.
The res argument is the response object with built-in methods to send data to the client-side through HTTP requests.
The argument, next, is a function that tells Express.js to continue on to the following middleware you have configured for your application.

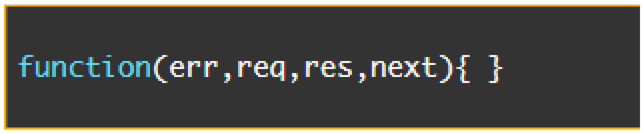
### What is ‘next’ in ExpressJs? And How it works ?

‘next’ is a parameter that is passed in middleware function to pass control to the next middleware,(if next middleware exists) else it will take to the api from which call is come to the server .



### Can middleware be able for error handling, Explain error handling in ExpressJs ?

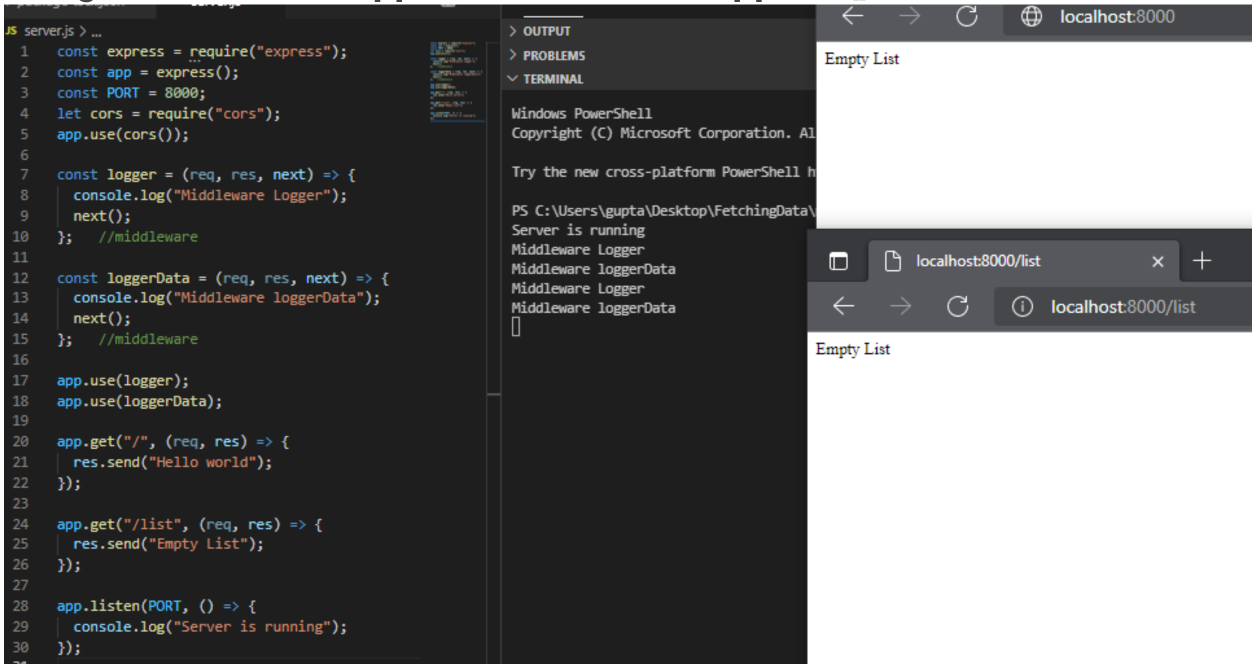
Middleware is able to handle error handling in ExpressJs. This can be done by passing extra error-handling parameter (err) in the middleware function.
Syntax .



```
function(err,req,res,next){ }
```

## What are different ways of using middleware?

### Using middleware at application level with app.use()



```javascript
const express = require("express");
const app = express();
const PORT = 8000;
let cors = require("cors");
app.use(cors());

const logger = (req, res, next) => {
  console.log("Middleware Logger");
  next();
};   //middleware

const loggerData = (req, res, next) => {
  console.log("Middleware loggerData");
  next();
};   //middleware

app.use(logger);
app.use(loggerData);

app.get("/", (req, res) => {
  res.send("Hello world");
});

app.get("/list", (req, res) => {
  res.send("Empty List");
});

app.listen(PORT, () => {
  console.log("Server is running");
});
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Al

Try the new cross-platform PowerShell h

PS C:\Users\gupta\Desktop\FetchingData\
Server is running
Middleware Logger
Middleware loggerData
Middleware Logger
Middleware loggerData
```

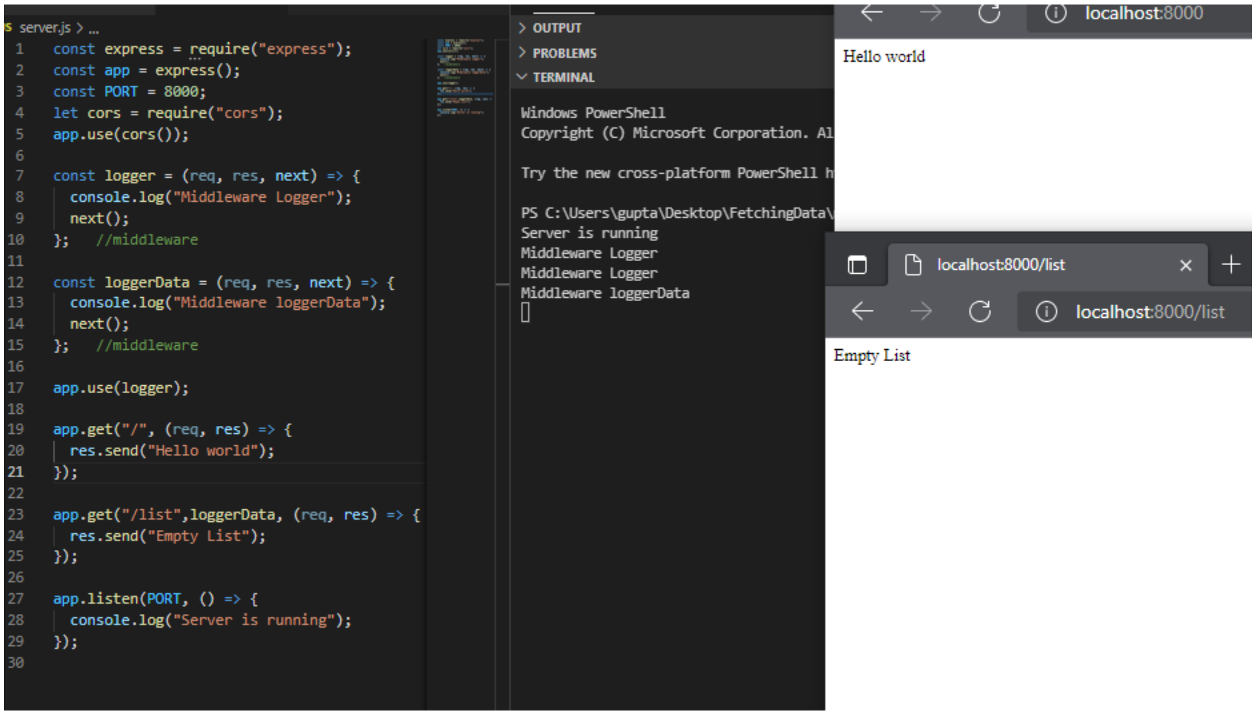
localhost:8000

Empty List

localhost:8000/list

Empty List

### Router Level Middleware



```javascript
const express = require("express");
const app = express();
const PORT = 8000;
let cors = require("cors");
app.use(cors());

const logger = (req, res, next) => {
  console.log("Middleware Logger");
  next();
};   //middleware

const loggerData = (req, res, next) => {
  console.log("Middleware loggerData");
  next();
};   //middleware

app.use(logger);

app.get("/", (req, res) => {
  res.send("Hello world");
});

app.get("/list",loggerData, (req, res) => {
  res.send("Empty List");
});

app.listen(PORT, () => {
  console.log("Server is running");
});
```

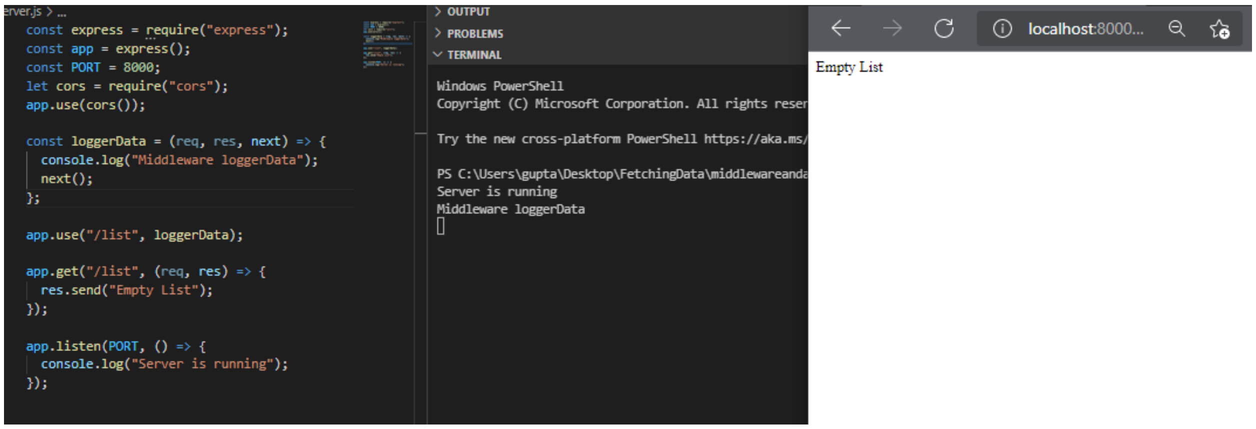
```
Windows PowerShell
Copyright (C) Microsoft Corporation. Al

Try the new cross-platform PowerShell h

PS C:\Users\gupta\Desktop\FetchingData\
Server is running
Middleware Logger
Middleware Logger
Middleware loggerData
```

localhost:8000

Hello world

localhost:8000/list

Empty List



```
const express = require("express");
const app = express();
const PORT = 8000;
let cors = require("cors");
app.use(cors());

const loggerData = (req, res, next) => {
  console.log("Middleware loggerData");
  next();
};

app.use("/list", loggerData);

app.get("/list", (req, res) => {
  res.send("Empty List");
});

app.listen(PORT, () => {
  console.log("Server is running");
});
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reser

Try the new cross-platform PowerShell https://aka.ms/

PS C:\Users\gupta\Desktop\FetchingData\middlewareanda
Server is running
Middleware loggerData
```

Empty List

II ROUTING MIDDLEWARE

What is Express.Router() ?

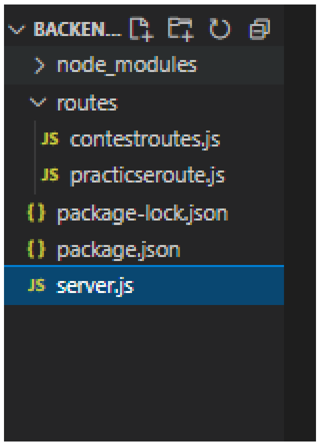
It is a function used to create a new router object. This function is used when we want to create a new router object in our program to handle requests .

**Using middleware we can also redirect our request to some pages**

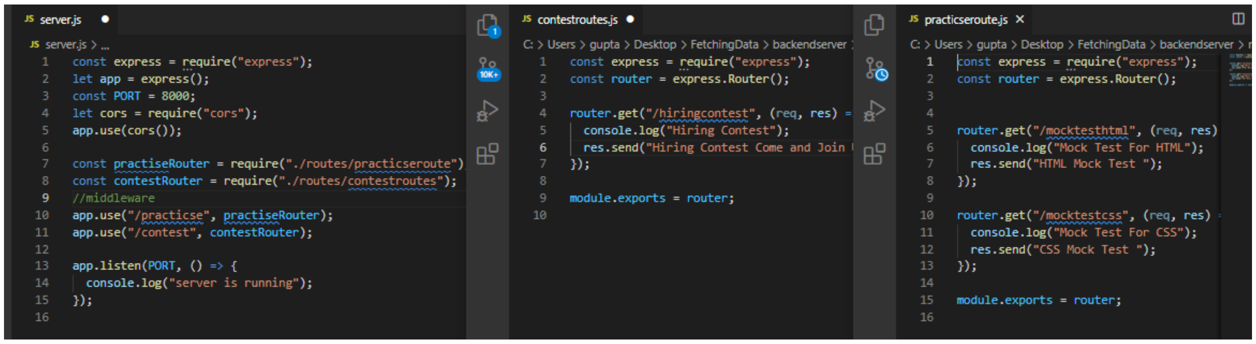
When we have to handle a lot of routings in our server.js due to which in server.js there will be a too much of code ,To get rid out off these we can create individual files and then using Middleware in our server.js we can reach to individual created files(or that particular route)

## How to use Middleware to redirect requests to other Pages ?

**Folder Structure**



**Files : server.js,contestroutes.js,practicseroute.js**


server.js

```
const express = require("express");
let app = express();
const PORT = 8000;
let cors = require("cors");
app.use(cors());

const practiseRouter = require("./routes/practicseroute")
const contestRouter = require("./routes/contestroutes");
//middleware
app.use("/practicse", practiseRouter);
app.use("/contest", contestRouter);

app.listen(PORT, () => {
  console.log("server is running");
});
```

contestroutes.js

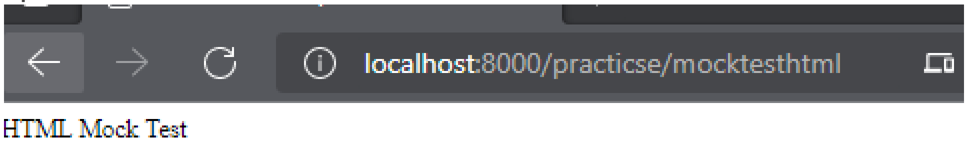
```
const express = require("express");
const router = express.Router();

router.get("/hiringcontest", (req, res) =
  console.log("Hiring Contest");
  res.send("Hiring Contest Come and Join
});

module.exports = router;
```

practicseroute.js

```
const express = require("express");
const router = express.Router();


router.get("/mocktesthtml", (req, res)
  console.log("Mock Test For HTML");
  res.send("HTML Mock Test ");
});

router.get("/mocktestcss", (req, res)
  console.log("Mock Test For CSS");
  res.send("CSS Mock Test ");
});

module.exports = router;
```

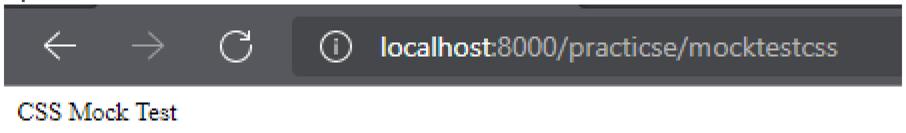
**Output**

/practicse/mocktesthtml


localhost:8000/practicse/mocktesthtml

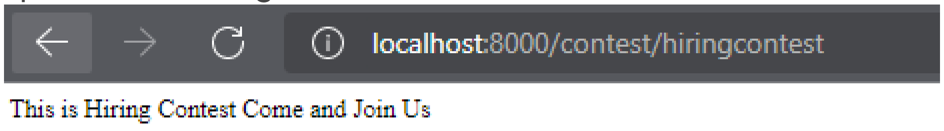
HTML Mock Test

/practicse/mocktestcss


localhost:8000/practicse/mocktestcss

CSS Mock Test

/practicse/hiringcontest


localhost:8000/contest/hiringcontest

This is Hiring Contest Come and Join Us

III HASHING PASSWORD

## What is Hashing Password?

Hashing turns your password (or any other piece of data) into a short string of letters or numbers using an encryption algorithm. If a website is hacked, the hackers don't get access to your password

## What is bcrypt?

Bcrypt is a popular and trusted method for salt and hashing passwords

## What are encryption and decryption?

Encryption is the process of translating plain text data (plaintext) into something that appears to be random and meaningless (ciphertext).

Decryption is the process of converting ciphertext back to plaintext.

**Different methods of Hashing password :** PBKDF2, bcrypt or scrypt,

## What is Salt in bcrypt ?

A salt is a random string that makes the hash unpredictable.

## Why do we use bcrypt?

Faster the algorithm is, Faster to break the password
Since BCRYPT is a slow algorithm, it has decryption slowly . Hence hackers will not crack password easily .

Installation bcrypt

```
  npm i bcrypt
```

## What is salt Round in bcrypt?

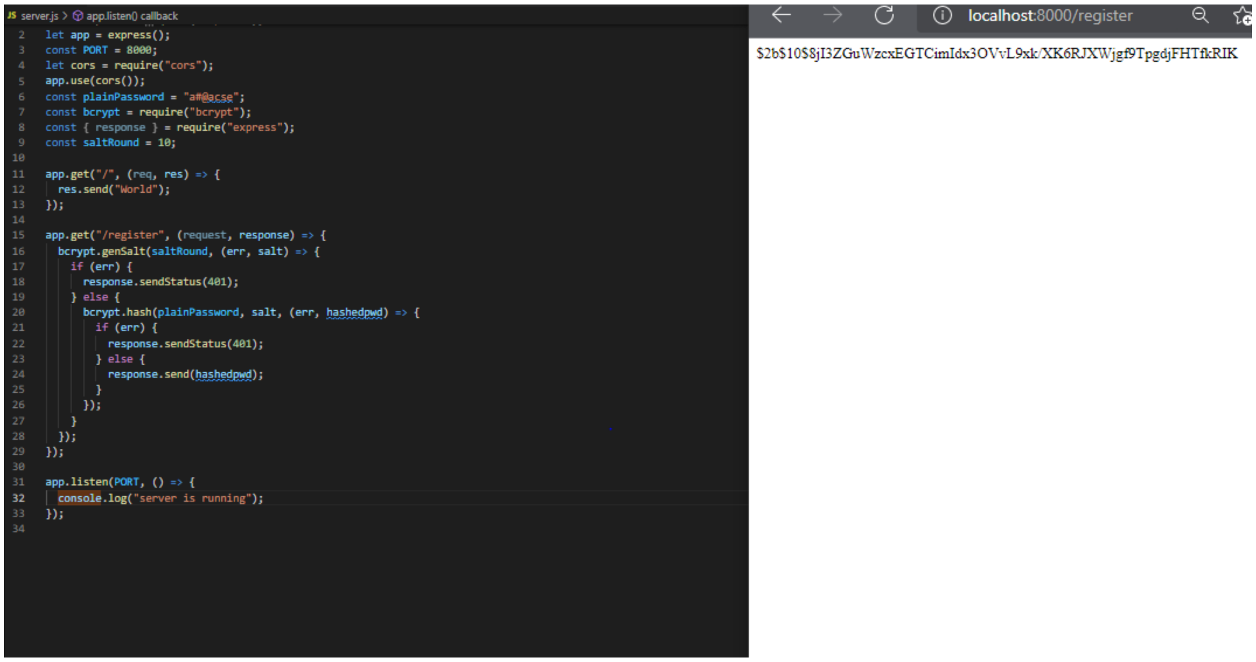
"salt round" actually mean the cost factor.
The cost factor controls how much time is needed to calculate a single bcrypt hash.
The higher the cost factor, the more hashing rounds are done. Increasing the cost factor by 1 double the necessary time.

## What are different ways to hash password through bcrypt?

There are two ways to hash password through bcrypt

1. Generate salt and then Generate hashed password
2. Generate hashed and password together

**Generate Salt and then generate password**

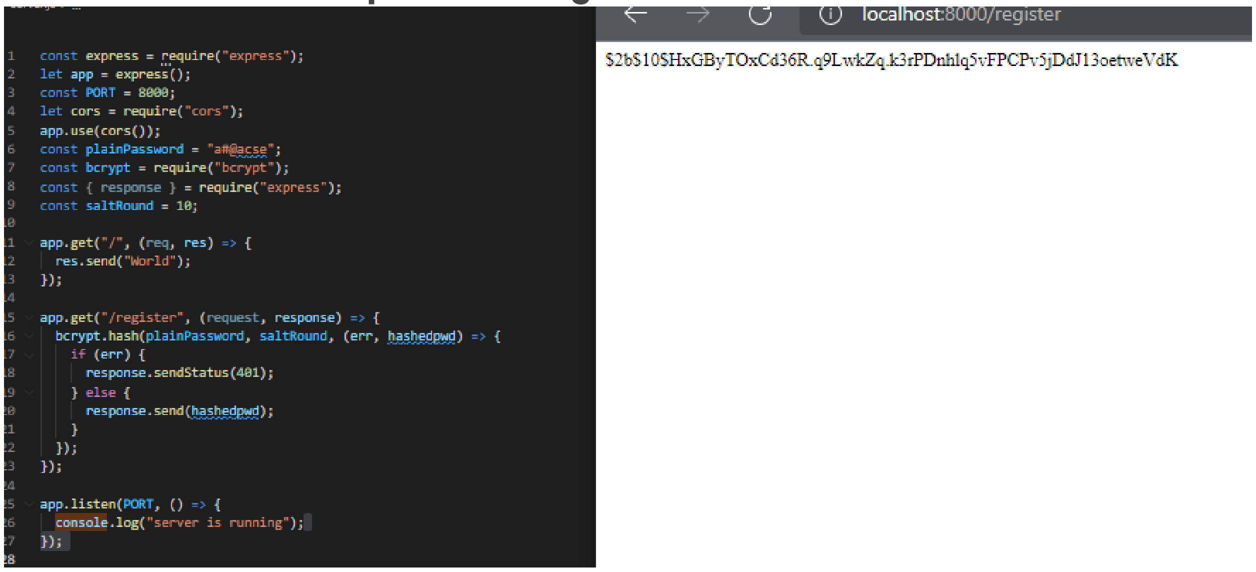


```
let app = express();
const PORT = 8000;
let cors = require("cors");
app.use(cors());
const plainPassword = "a#@acse";
const bcrypt = require("bcrypt");
const { response } = require("express");
const saltRound = 10;

app.get("/", (req, res) => {
  res.send("World");
});

app.get("/register", (request, response) => {
  bcrypt.genSalt(saltRound, (err, salt) => {
    if (err) {
      response.sendStatus(401);
    } else {
      bcrypt.hash(plainPassword, salt, (err, hashedpwd) => {
        if (err) {
          response.sendStatus(401);
        } else {
          response.send(hashedpwd);
        }
      });
    }
  });
});

app.listen(PORT, () => {
  console.log("server is running");
});
```

localhost:8000/register

```
$2b$10$8jI3ZGuWzcxEGTCimIdx3OVvL9xk/XK6RJXWjgf9TpgdjFHTfkRIK
```

## Generate hashed and password together



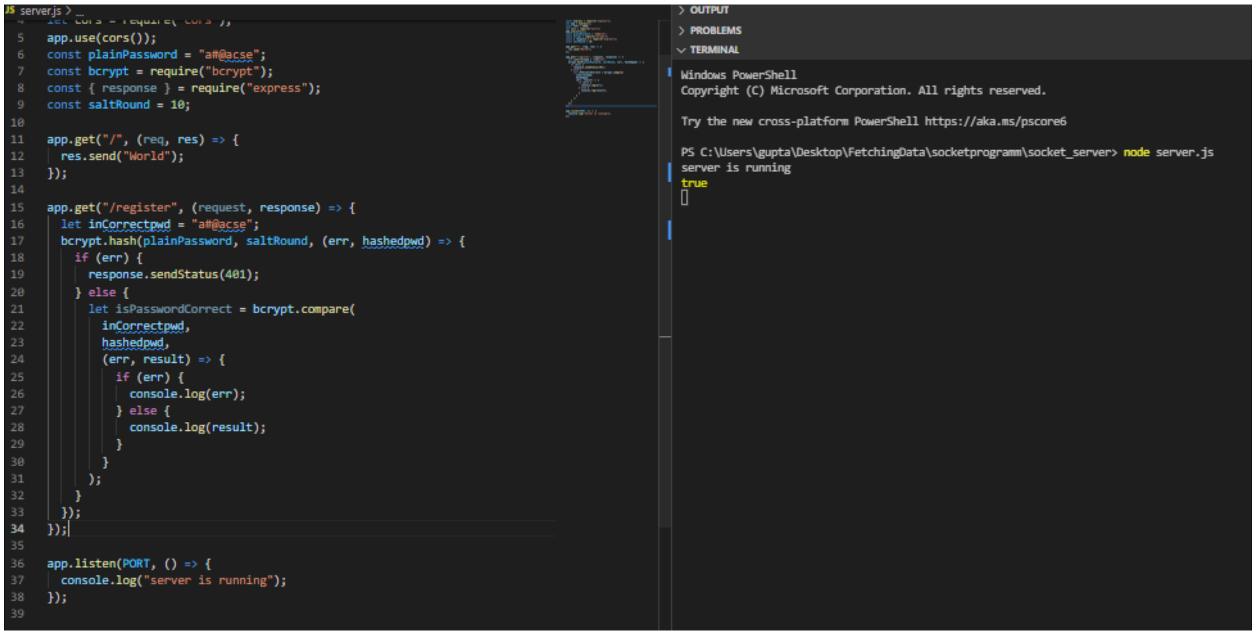
```
const express = require("express");
let app = express();
const PORT = 8000;
let cors = require("cors");
app.use(cors());
const plainPassword = "a#@acse";
const bcrypt = require("bcrypt");
const { response } = require("express");
const saltRound = 10;

app.get("/", (req, res) => {
  res.send("World");
});

app.get("/register", (request, response) => {
  bcrypt.hash(plainPassword, saltRound, (err, hashedpwd) => {
    if (err) {
      response.sendStatus(401);
    } else {
      response.send(hashedpwd);
    }
  });
});

app.listen(PORT, () => {
  console.log("server is running");
});
```

localhost:8000/register

```
$2b$10$HxGByTOxCd36R.q9LwkZq.k3rPDnhlq5vFPCPv5jDdJ13oetweVdK
```

## What is bcrypt.compare()?

This method compares hashed and plaintext passwords without the salt string



server.js > ...

```javascript
let cors = require("cors");
app.use(cors());
const plainPassword = "a#@acse";
const bcrypt = require("bcrypt");
const { response } = require("express");
const saltRound = 10;

app.get("/", (req, res) => {
  res.send("World");
});

app.get("/register", (request, response) => {
  let inCorrectpwd = "a#@acse";
  bcrypt.hash(plainPassword, saltRound, (err, hashedpwd) => {
    if (err) {
      response.sendStatus(401);
    } else {
      let isPasswordCorrect = bcrypt.compare(
        inCorrectpwd,
        hashedpwd,
        (err, result) => {
          if (err) {
            console.log(err);
          } else {
            console.log(result);
          }
        }
      );
    }
  });
});

app.listen(PORT, () => {
  console.log("server is running");
});
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\gupta\Desktop\FetchingData\socketprogramm\socket_server> node server.js
server is running
true
```

# Socket Programming

## I. INTRODUCTION TO SOCKET PROGRAMMING

### What is Socket Programming?

A socket Programming is a endpoint of two way Communication link between two programs running on the network . A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to .

### What is tcp?

TCP stands for Transmission Control Protocol a communications standard that enables application programs and computing devices to exchange messages over a network
OR
TCP is one of the basic standards that define the rules of the internet and is included within the standards defined by the Internet Engineering Task Force (IETF).

### What is IP?

Every device has an IP address The Internet Protocol (IP) is the method for sending data from one device to another across the internet. Every device has an IP address that uniquely identifies it and enables it to communicate with and exchange data with other devices connected to the internet.

## What is the difference between http connection and webSocket Connection?

**Http connection :**

1. It is unidirectional where client sends request and server send response
2. Example client send the request , corresponding to that request server send the response , after sending the response connection get terminated
3. If a user is sending 100 requests then corresponding to each request server sends a response and each http and https request establishes the new connection to the server every time and after getting the response the connection gets terminated by itself.
4. **Stateless Protocol:** Stateless Protocols are the type of network protocols in which the Client sends a request to the server and the server response back according to the current state

**WebSocket Connection :**

1. It is a bidirectional full-duplex protocol.
2. whenever we initiate the connection between client server the client-server made the handshaking and decide to create a new connection and this connection will keep alive until terminated by any of them.

3. In websocket conection , connection will not terminate after a single request response
4. **Stateful Protocol:** A Stateful Protocol is a type of network protocol in which the client sends a server request and expects some sort of response. In case it doesn't get a response, it then resends the intended request

### Where do we use Socket Programming ?

It is used in various Apps Like Chat App , Game App , Notification Engine

### What are the advantages and Disadvantages of Socket Programming?

**Advantages:**

1. Socket is flexible and sufficient.
2. Efficient socket-based programming can be easily implemented for general communications.
3. It causes low network traffic.

**Disadvantages:**

1. Socket-based communication allows only to send packets of raw data between applications. Both the client-side and server-side have to provide mechanisms to make the data useful in any way.

# II BUILDING SOCKET CONNECTIONS

Installation

For building Socket Connections

```
Frontend - npm install socket.io-client
Backend - npm install socket.io
```

### What is Socket.io?

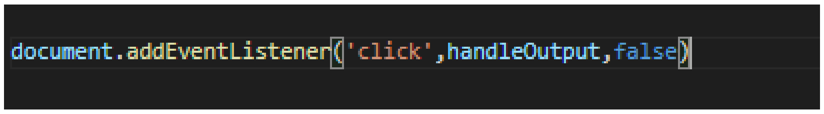
Socket.IO is a library that enables real-time, bidirectional, and event-based communication between the browser and the server.

### What is Socket in Socket.io?

A Socket is basically an EventEmitter that sends events to — and receives events from — the server over the network.

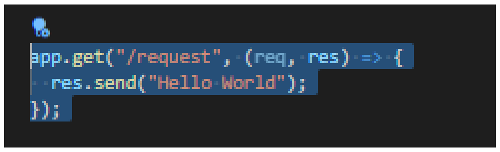
## What are event Listeners?

An event listener is a function in JavaScript that waits for an event to occur. And after the event occurred there will be a certain set of actions(Whatever we have defined in our code) executed.



```
document.addEventListener('click',handleOutput,false)
```

Event Listeners in Nodejs are the same as the callback. A callback function is called when a function execution is completed



```
app.get("/request", (req, res) => {
  res.send("Hello World");
});
```

Whenever we hit /request on the browser, there will be a callback, according to /request response, we will send the response to our frontend
also, event listeners work when any specified event has fired

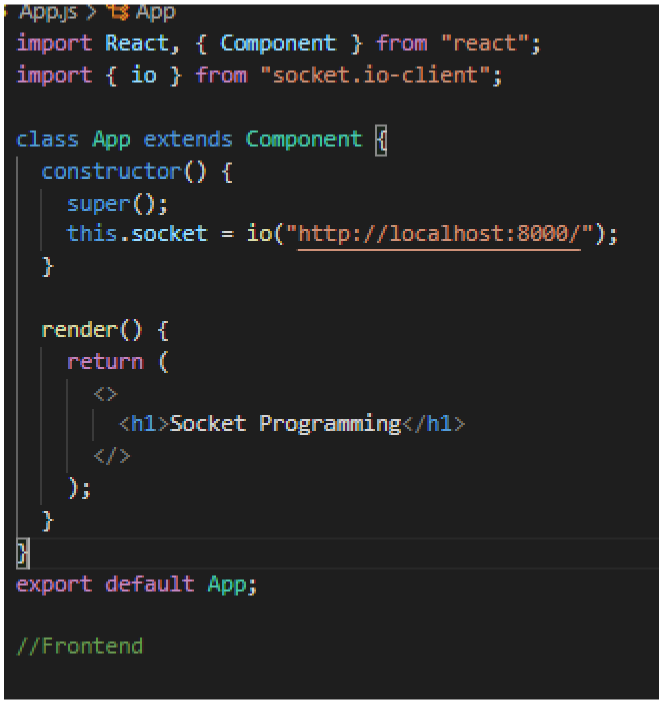
## List Some EventEmitter methods ?

**socket.emit -** This method is responsible for sending messages.
**socket.on -** This method is responsible for listening for incoming messages.
**socket. off(eventName, listener) -** Removes the specified listener from the listener array for the event named eventName.
**socket. once(event name, event listener) -** Adds a one-time listener for the event, after which it is removed. This listener is invoked only the next time the event is fired.
**socket.removeAllListeners(            ) -** Removes all listeners.

## Code For Setting Connection between Client Side and Server-side in socket Programming

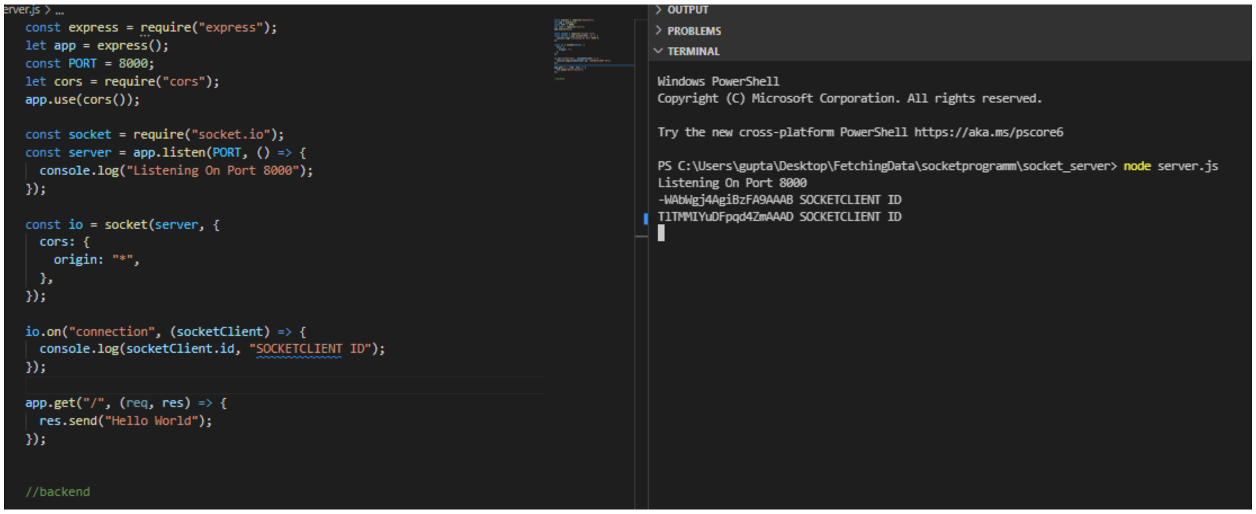
The frontend and backend will run on a different port

## Frontend



```javascript
import React, { Component } from "react";
import { io } from "socket.io-client";

class App extends Component {
  constructor() {
    super();
    this.socket = io("http://localhost:8000/");
  }

  render() {
    return (
      <>
        <h1>Socket Programming</h1>
      </>
    );
  }
}
export default App;

//Frontend
```

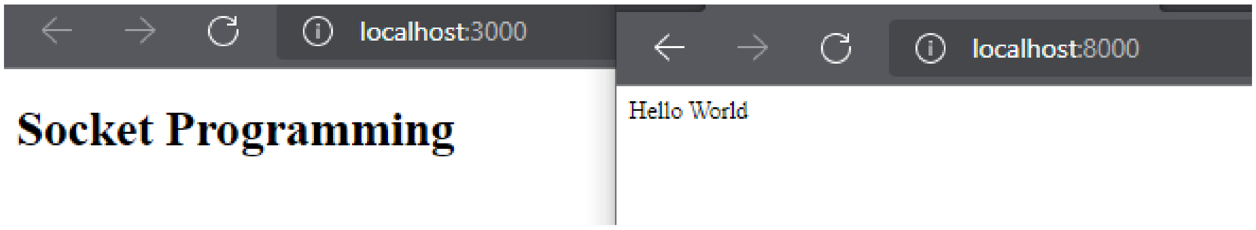
## Backend



```javascript
const express = require("express");
let app = express();
const PORT = 8000;
let cors = require("cors");
app.use(cors());

const socket = require("socket.io");
const server = app.listen(PORT, () => {
  console.log("Listening On Port 8000");
});

const io = socket(server, {
  cors: {
    origin: "*",
  },
});

io.on("connection", (socketClient) => {
  console.log(socketClient.id, "SOCKETCLIENT ID");
});

app.get("/", (req, res) => {
  res.send("Hello World");
});


//backend
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\gupta\Desktop\FetchingData\socketprogramm\socket_server> node server.js
Listening On Port 8000
-WAbWgj4AgiBzFA9AAAB SOCKETCLIENT ID
T1TMMIYuDFpqd4ZmAAAD SOCKETCLIENT ID
```

## Connections



Socket Programming

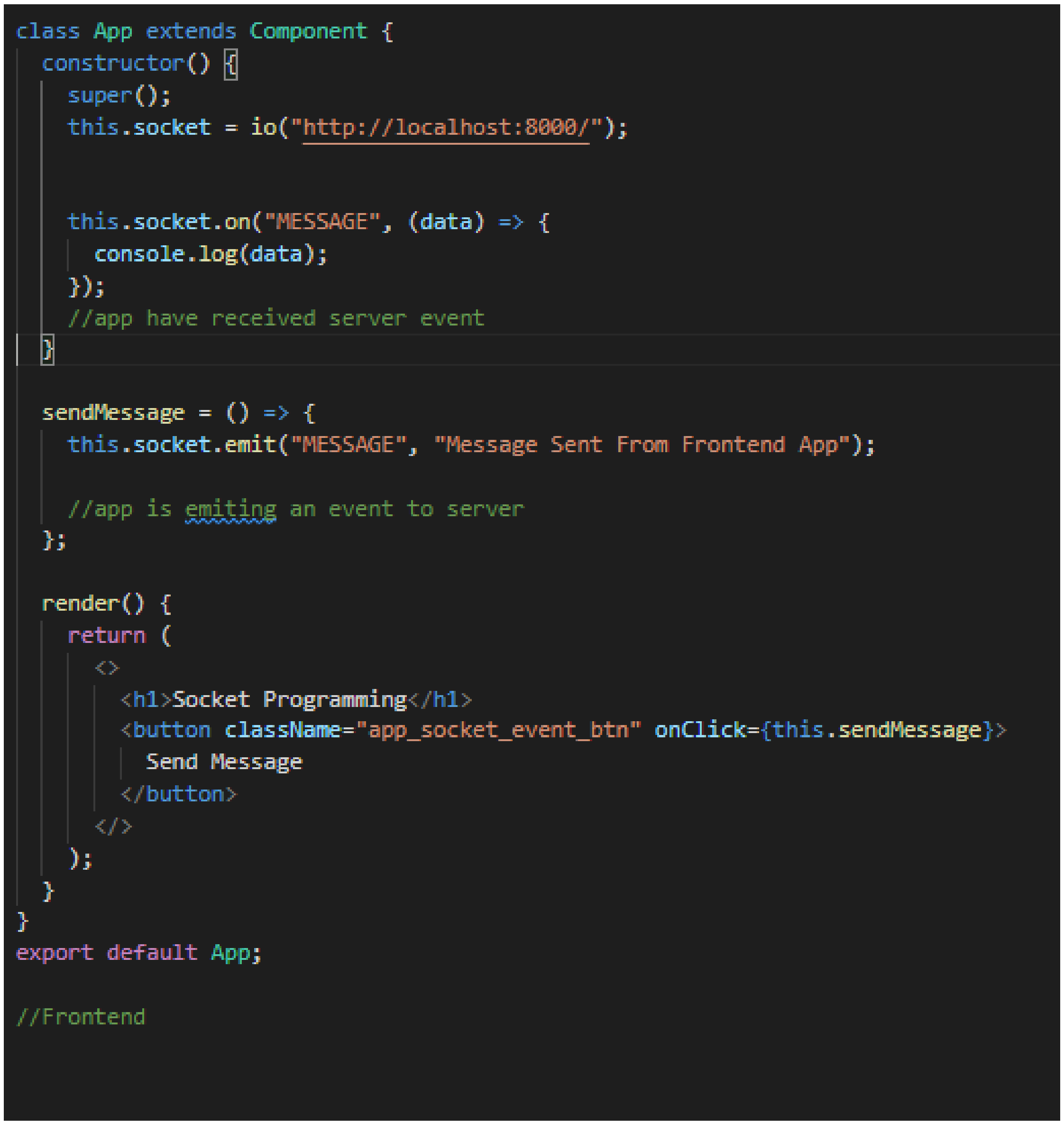
Hello World

## III EVENTS IN SOCKET PROGRAMMING

- The Interactions between client and server happen through events .
- One party emit an event and another party will listen .
- And another party do the actions whatsoever the event emitted .

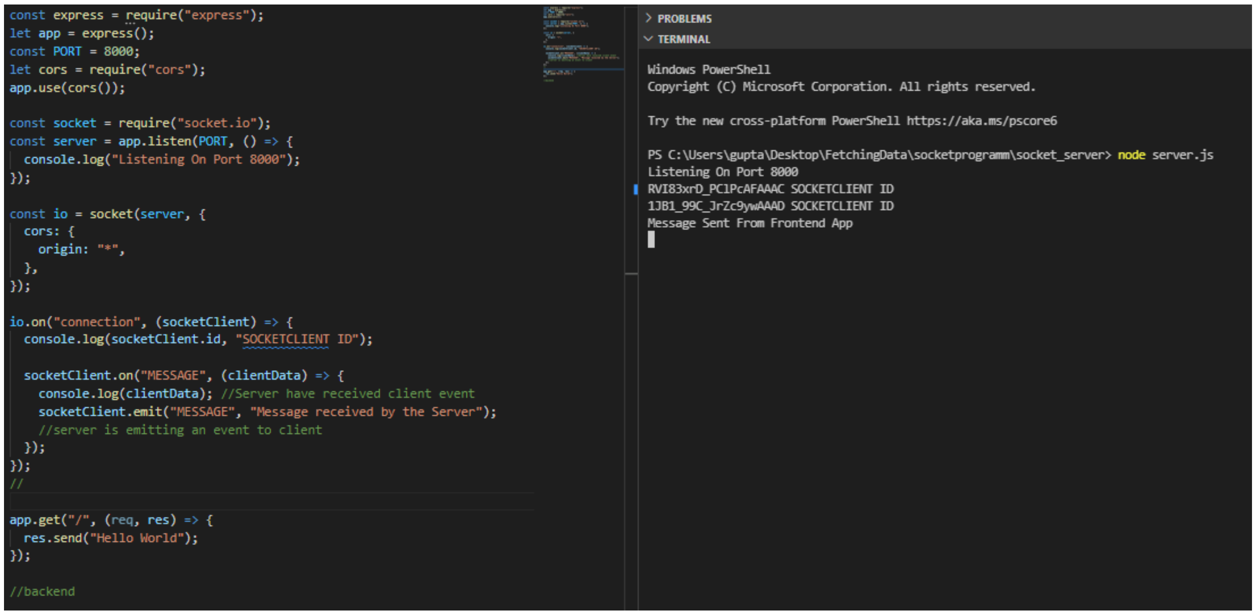
**Case when Frontend (Client) and backend (server) both are emitting event**

**Frontend Code**



```
class App extends Component {
  constructor() {
    super();
    this.socket = io("http://localhost:8000/");


    this.socket.on("MESSAGE", (data) => {
      console.log(data);
    });
    //app have received server event
  }

  sendMessage = () => {
    this.socket.emit("MESSAGE", "Message Sent From Frontend App");

    //app is emiting an event to server
  };

  render() {
    return (
      <>
        <h1>Socket Programming</h1>
        <button className="app_socket_event_btn" onClick={this.sendMessage}>
          Send Message
        </button>
      </>
    );
  }
}
export default App;

//Frontend
```

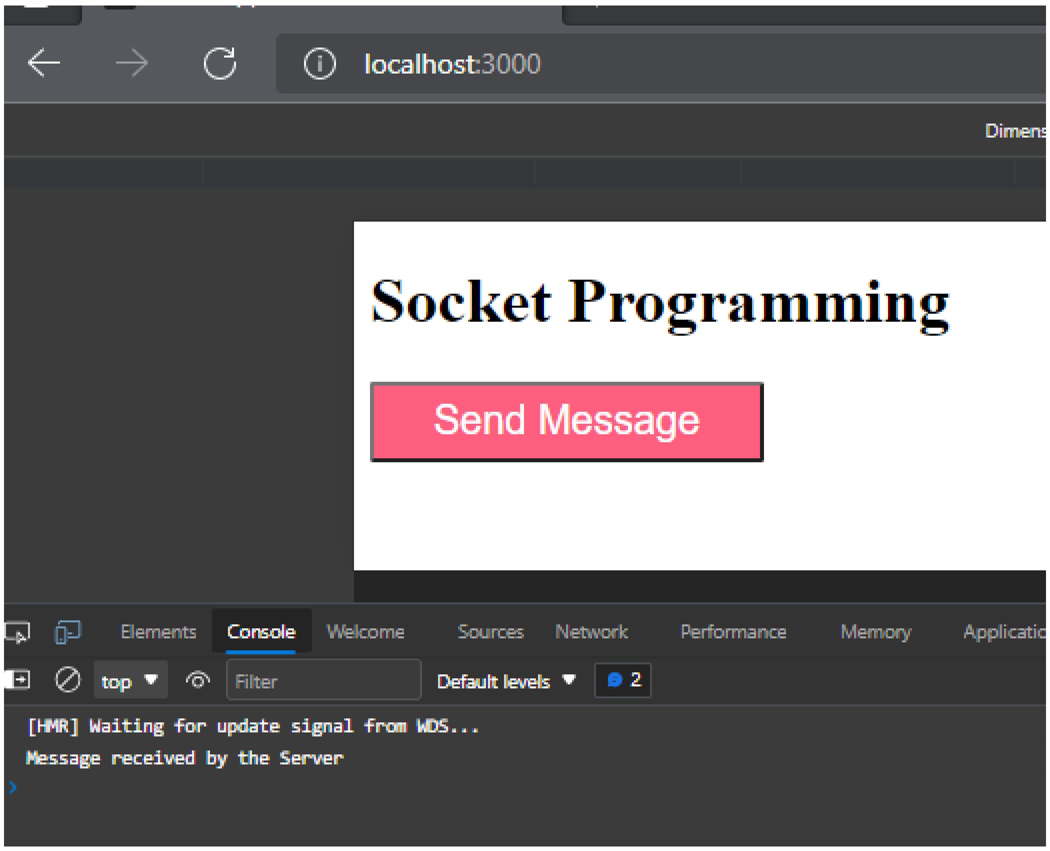
## Backend Code



```javascript
const express = require("express");
let app = express();
const PORT = 8000;
let cors = require("cors");
app.use(cors());

const socket = require("socket.io");
const server = app.listen(PORT, () => {
  console.log("Listening On Port 8000");
});

const io = socket(server, {
  cors: {
    origin: "*",
  },
});

io.on("connection", (socketClient) => {
  console.log(socketClient.id, "SOCKETCLIENT ID");

  socketClient.on("MESSAGE", (clientData) => {
    console.log(clientData); //Server have received client event
    socketClient.emit("MESSAGE", "Message received by the Server");
    //server is emitting an event to client
  });
});
//

app.get("/", (req, res) => {
  res.send("Hello World");
});

//backend
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\gupta\Desktop\FetchingData\socketprogramm\socket_server> node server.js
Listening On Port 8000
RVI83xrD_PC1PcAFAAAC SOCKETCLIENT ID
1JB1_99C_JrZc9ywAAAD SOCKETCLIENT ID
Message Sent From Frontend App
```

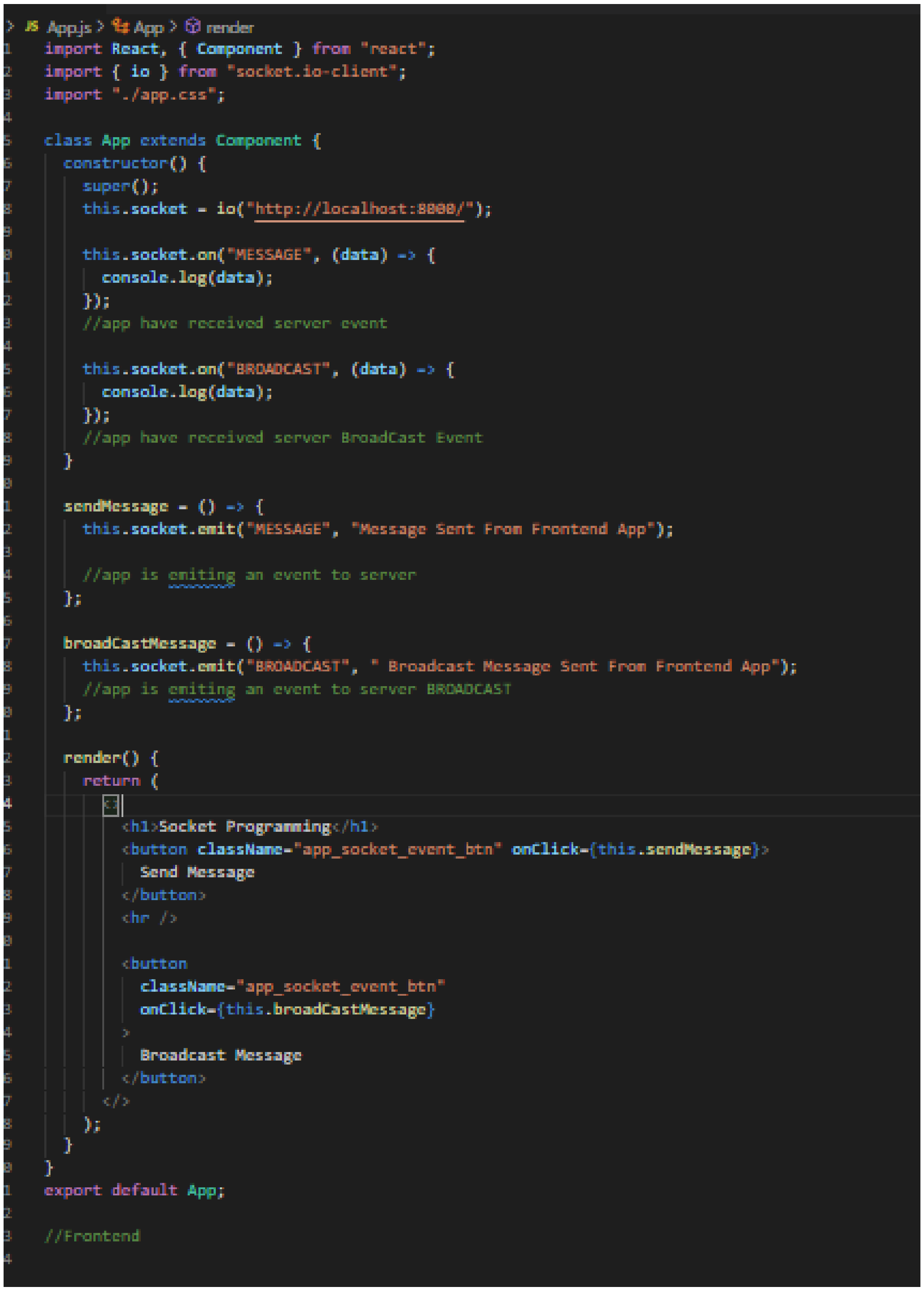
## Browser



```
[HMR] Waiting for update signal from WDS...
Message received by the Server
```

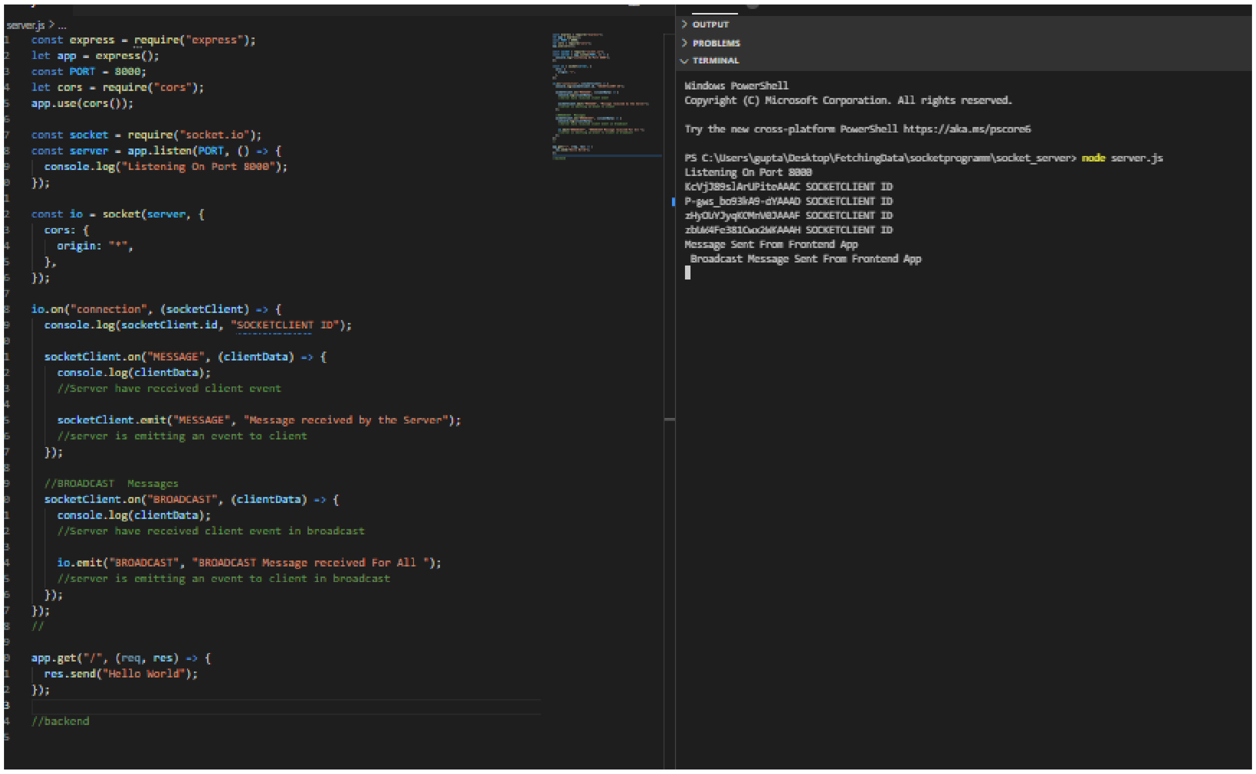
**Case : Broadcast Message When all the clients connected to the server received the server event**

**Frontend Code**



```
import React, { Component } from "react";
import { io } from "socket.io-client";
import "./app.css";

class App extends Component {
  constructor() {
    super();
    this.socket = io("http://localhost:8080/");

    this.socket.on("MESSAGE", (data) => {
      console.log(data);
    });
    //app have received server event

    this.socket.on("BROADCAST", (data) => {
      console.log(data);
    });
    //app have received server BroadCast Event
  }

  sendMessage = () => {
    this.socket.emit("MESSAGE", "Message Sent From Frontend App");

    //app is emiting an event to server
  };

  broadCastMessage = () => {
    this.socket.emit("BROADCAST", " Broadcast Message Sent From Frontend App");
    //app is emiting an event to server BROADCAST
  };

  render() {
    return (
      <>
        <h1>Socket Programming</h1>
        <button className="app_socket_event_btn" onClick={this.sendMessage}>
          Send Message
        </button>
        <hr />

        <button
          className="app_socket_event_btn"
          onClick={this.broadCastMessage}
        >
          Broadcast Message
        </button>
      </>
    );
  }
}
export default App;

//Frontend
```

## Backend Code

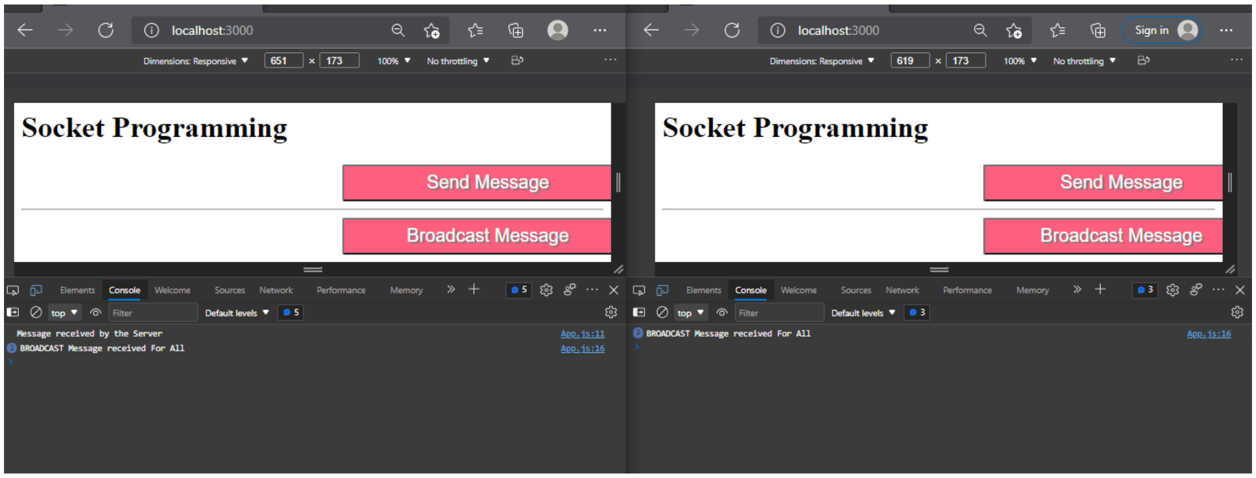


server.js > ...

```javascript
const express = require("express");
let app = express();
const PORT = 8000;
let cors = require("cors");
app.use(cors());

const socket = require("socket.io");
const server = app.listen(PORT, () => {
  console.log("Listening On Port 8000");
});

const io = socket(server, {
  cors: {
    origin: "*",
  },
});

io.on("connection", (socketClient) => {
  console.log(socketClient.id, "SOCKETCLIENT ID");

  socketClient.on("MESSAGE", (clientData) => {
    console.log(clientData);
    //Server have received client event

    socketClient.emit("MESSAGE", "Message received by the Server");
    //server is emitting an event to client
  });

  //BROADCAST  Messages
  socketClient.on("BROADCAST", (clientData) => {
    console.log(clientData);
    //Server have received client event in broadcast

    io.emit("BROADCAST", "BROADCAST Message received For All ");
    //server is emitting an event to client in broadcast
  });
});
//

app.get("/", (req, res) => {
  res.send("Hello World");
});

//backend
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\gupta\Desktop\FetchingData\socketprogramm\socket_server> node server.js
Listening On Port 8000
KcVjJ89slArUPiteAAAC SOCKETCLIENT ID
P-gws_bo93kA9-dYAAAD SOCKETCLIENT ID
zHyOUYJyqKCMnV0JAAAF SOCKETCLIENT ID
zbUW4Fe381Cwx2WKAAAH SOCKETCLIENT ID
Message Sent From Frontend App
 Broadcast Message Sent From Frontend App
```

## Browser



localhost:3000
Dimensions: Responsive 651 × 173 100% No throttling
Socket Programming
Send Message
Broadcast Message
Elements Console Welcome Sources Network Performance Memory
top Filter Default levels 5

```
Message received by the Server                App.js:11
BROADCAST Message received For All            App.js:16
```

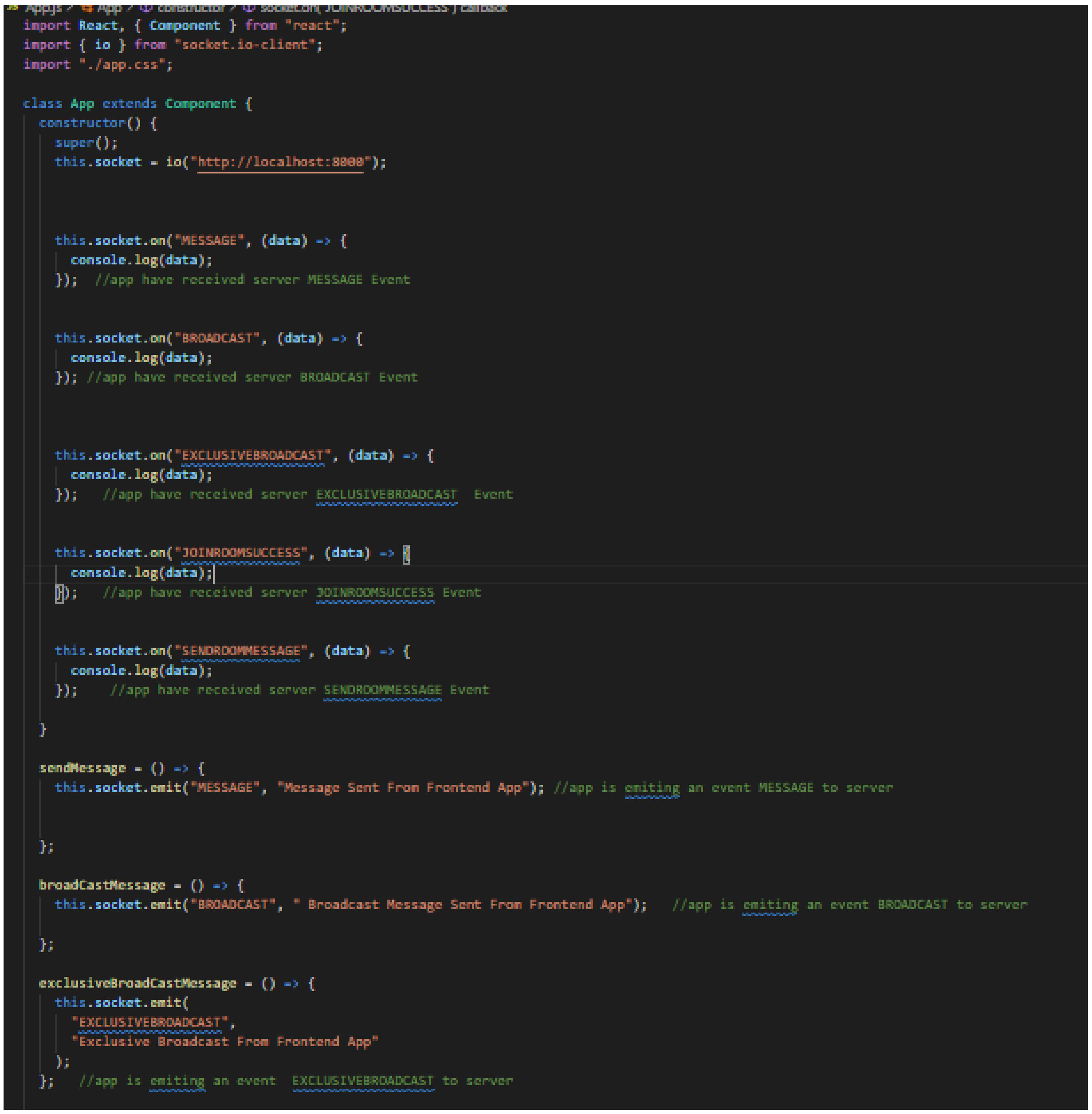
localhost:3000
Sign in
Dimensions: Responsive 619 × 173 100% No throttling
Socket Programming
Send Message
Broadcast Message
Elements Console Welcome Sources Network Performance Memory
top Filter Default levels 3

```
BROADCAST Message received For All            App.js:16
```

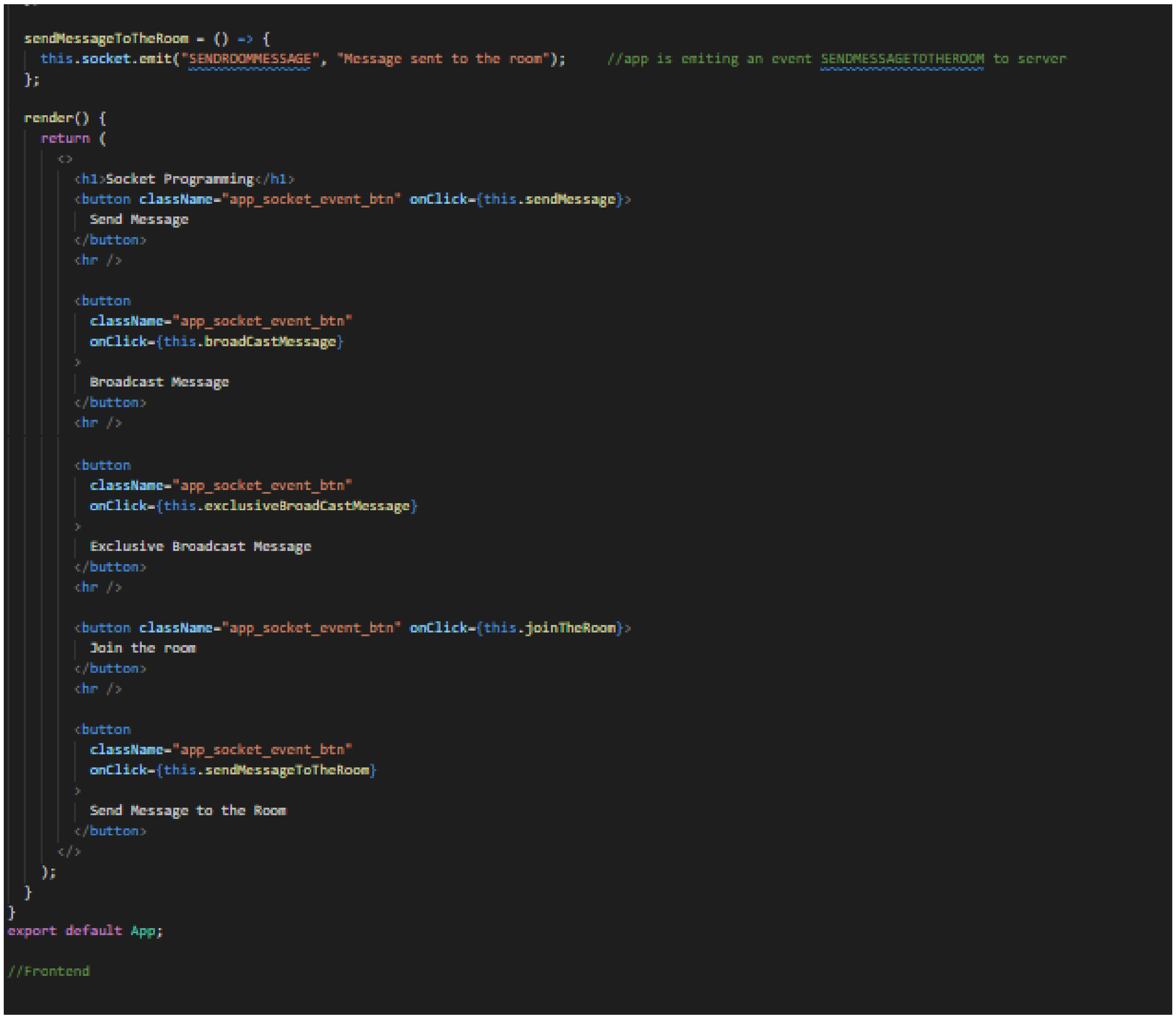
## IV ROOMS IN SOCKET PROGRAMMING

### Case: only those users who have joined the room received the message when any of the room member from the Room send the message(Group Chat)

### Frontend Code

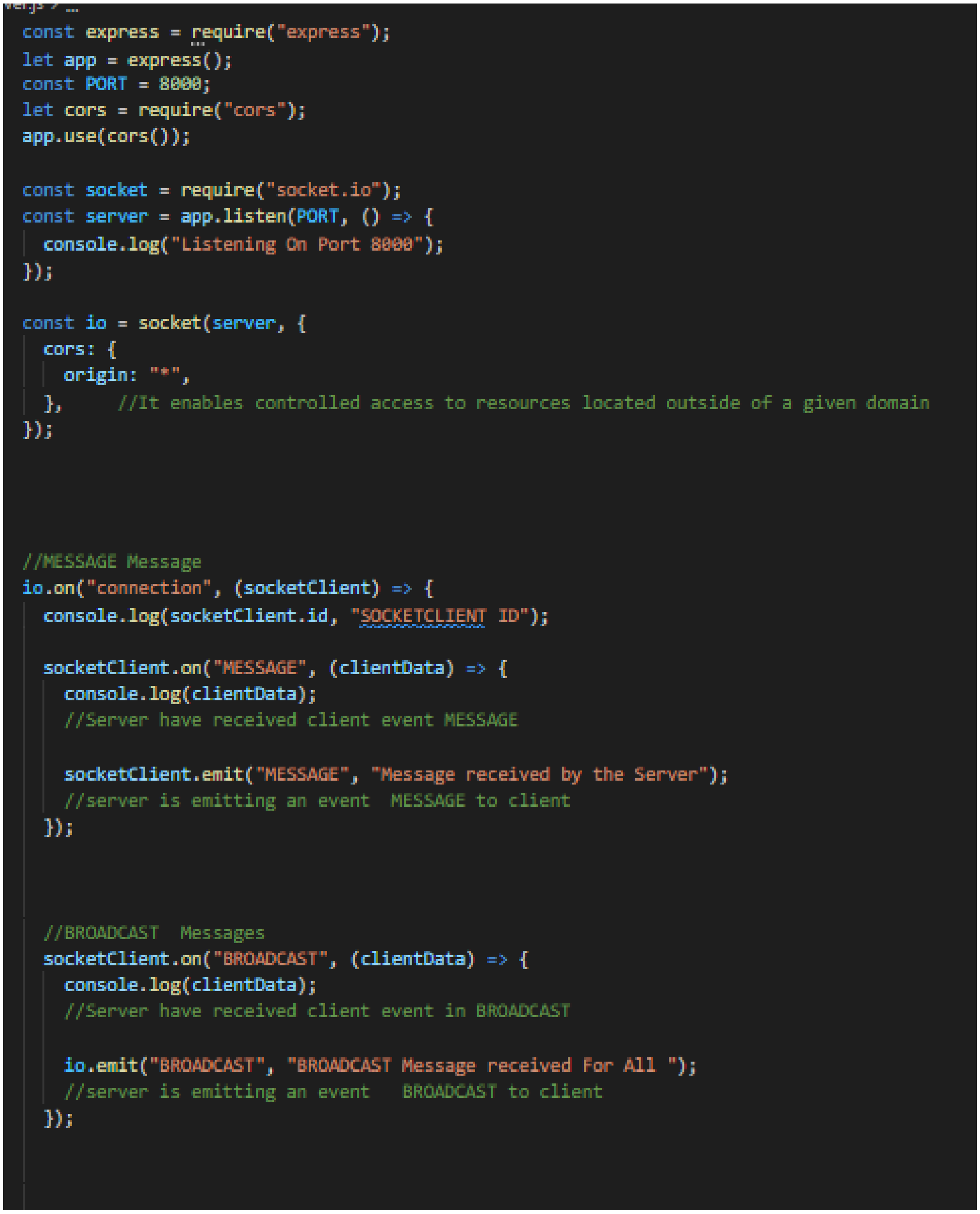


```
App.js > App > constructor > socket.on("JOINROOMSUCCESS") callback
import React, { Component } from "react";
import { io } from "socket.io-client";
import "./app.css";

class App extends Component {
  constructor() {
    super();
    this.socket = io("http://localhost:8000");



    this.socket.on("MESSAGE", (data) => {
      console.log(data);
    });  //app have received server MESSAGE Event


    this.socket.on("BROADCAST", (data) => {
      console.log(data);
    }); //app have received server BROADCAST Event



    this.socket.on("EXCLUSIVEBROADCAST", (data) => {
      console.log(data);
    });   //app have received server EXCLUSIVEBROADCAST  Event


    this.socket.on("JOINROOMSUCCESS", (data) => {
      console.log(data);
    });   //app have received server JOINROOMSUCCESS Event


    this.socket.on("SENDROOMMESSAGE", (data) => {
      console.log(data);
    });    //app have received server SENDROOMMESSAGE Event

  }

  sendMessage = () => {
    this.socket.emit("MESSAGE", "Message Sent From Frontend App"); //app is emiting an event MESSAGE to server


  };

  broadCastMessage = () => {
    this.socket.emit("BROADCAST", " Broadcast Message Sent From Frontend App");    //app is emiting an event BROADCAST to server

  };

  exclusiveBroadCastMessage = () => {
    this.socket.emit(
      "EXCLUSIVEBROADCAST",
      "Exclusive Broadcast From Frontend App"
    );
  };  //app is emiting an event  EXCLUSIVEBROADCAST to server
```

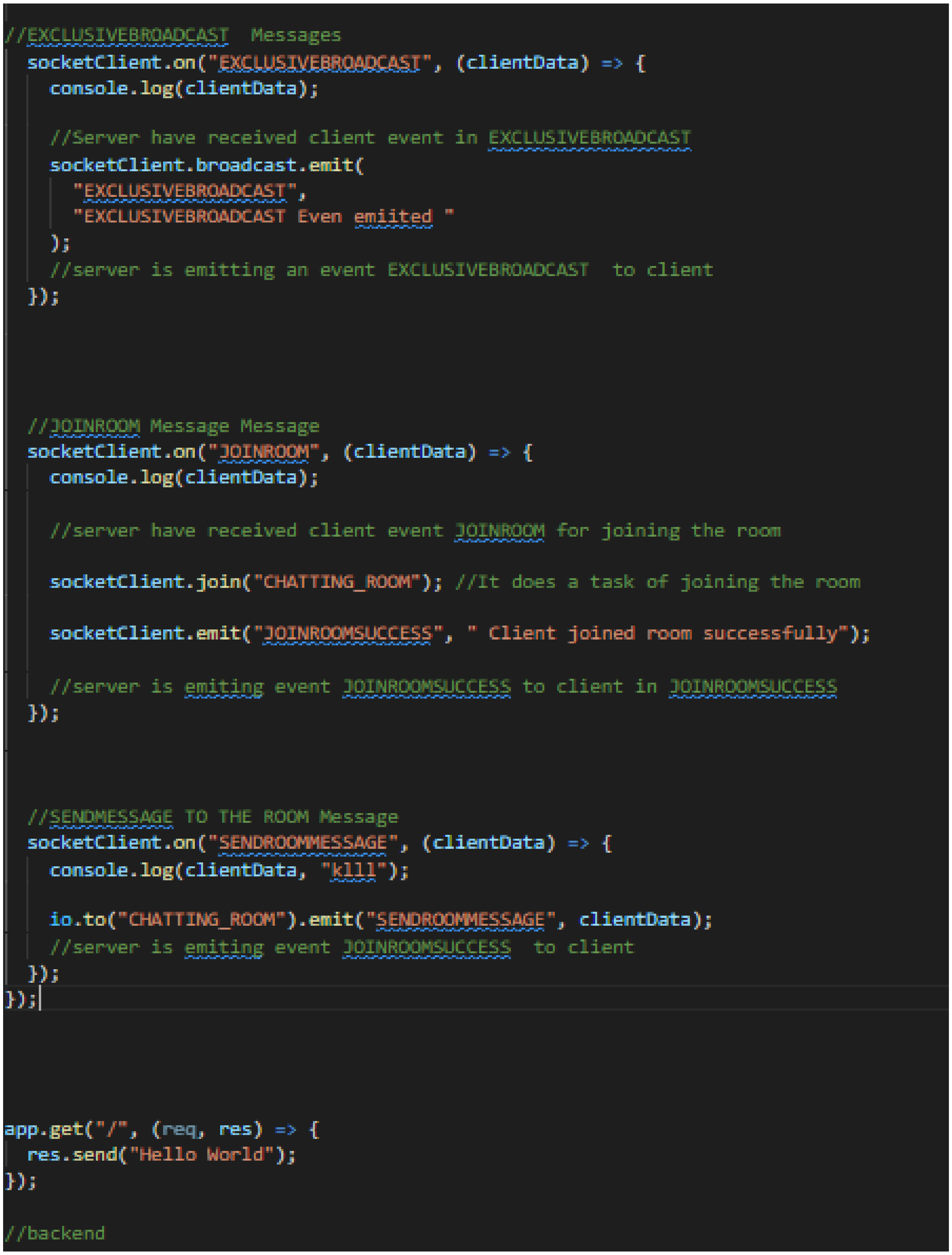


```
  sendMessageToTheRoom = () => {
    this.socket.emit("SENDROOMMESSAGE", "Message sent to the room");     //app is emiting an event SENDMESSAGETOTHEROOM to server
  };

  render() {
    return (
      <>
        <h1>Socket Programming</h1>
        <button className="app_socket_event_btn" onClick={this.sendMessage}>
          Send Message
        </button>
        <hr />

        <button
          className="app_socket_event_btn"
          onClick={this.broadCastMessage}
        >
          Broadcast Message
        </button>
        <hr />

        <button
          className="app_socket_event_btn"
          onClick={this.exclusiveBroadCastMessage}
        >
          Exclusive Broadcast Message
        </button>
        <hr />

        <button className="app_socket_event_btn" onClick={this.joinTheRoom}>
          Join the room
        </button>
        <hr />

        <button
          className="app_socket_event_btn"
          onClick={this.sendMessageToTheRoom}
        >
          Send Message to the Room
        </button>
      </>
    );
  }
}
export default App;

//Frontend
```

## Backend Code

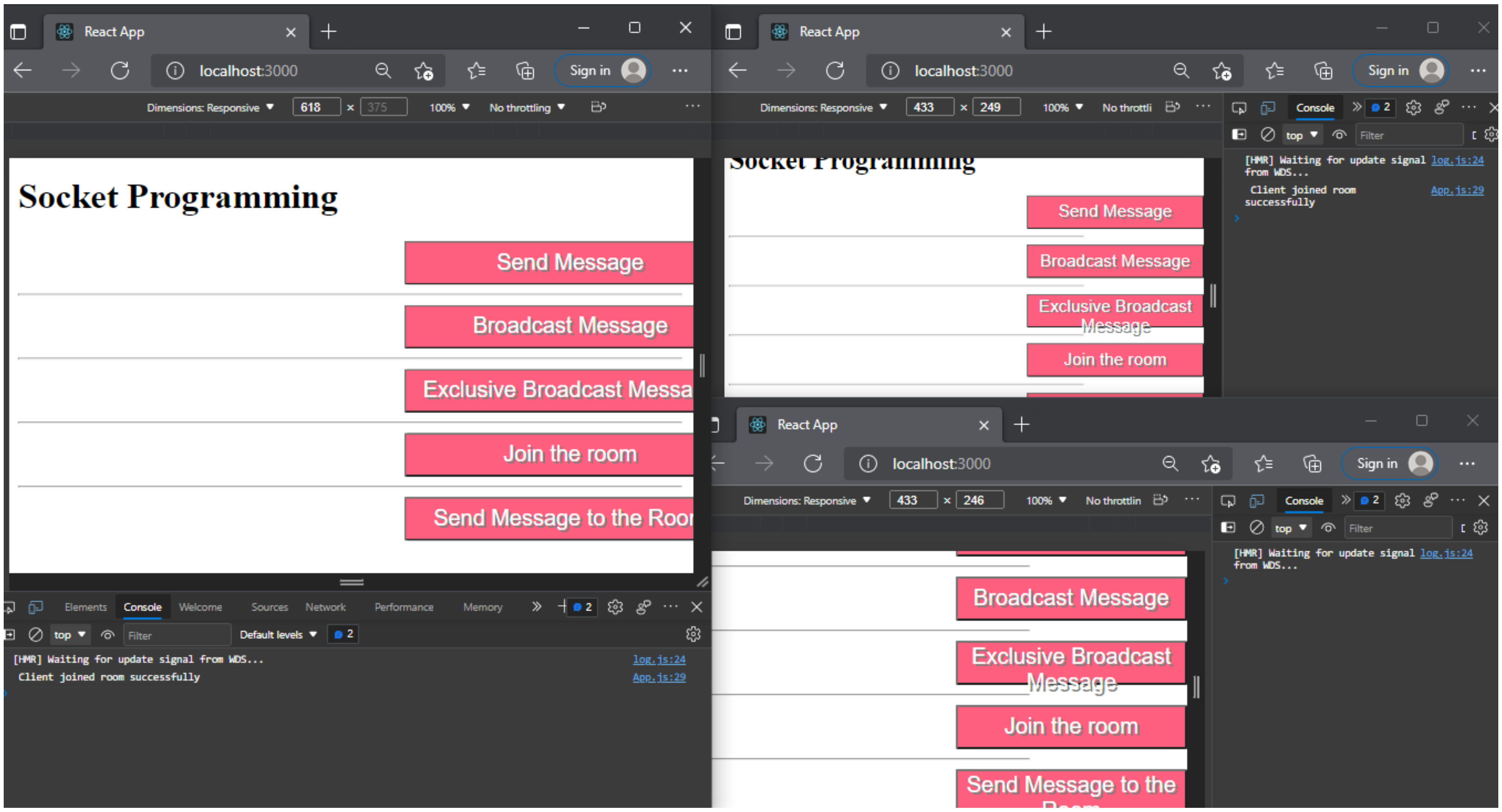


```
const express = require("express");
let app = express();
const PORT = 8000;
let cors = require("cors");
app.use(cors());

const socket = require("socket.io");
const server = app.listen(PORT, () => {
  console.log("Listening On Port 8000");
});

const io = socket(server, {
  cors: {
    origin: "*",
  },     //It enables controlled access to resources located outside of a given domain
});




//MESSAGE Message
io.on("connection", (socketClient) => {
  console.log(socketClient.id, "SOCKETCLIENT ID");

  socketClient.on("MESSAGE", (clientData) => {
    console.log(clientData);
    //Server have received client event MESSAGE

    socketClient.emit("MESSAGE", "Message received by the Server");
    //server is emitting an event  MESSAGE to client
  });



  //BROADCAST  Messages
  socketClient.on("BROADCAST", (clientData) => {
    console.log(clientData);
    //Server have received client event in BROADCAST

    io.emit("BROADCAST", "BROADCAST Message received For All ");
    //server is emitting an event   BROADCAST to client
  });
```



```
//EXCLUSIVEBROADCAST  Messages
  socketClient.on("EXCLUSIVEBROADCAST", (clientData) => {
    console.log(clientData);

    //Server have received client event in EXCLUSIVEBROADCAST
    socketClient.broadcast.emit(
      "EXCLUSIVEBROADCAST",
      "EXCLUSIVEBROADCAST Even emiited "
    );
    //server is emitting an event EXCLUSIVEBROADCAST  to client
  });



  //JOINROOM Message Message
  socketClient.on("JOINROOM", (clientData) => {
    console.log(clientData);

    //server have received client event JOINROOM for joining the room

    socketClient.join("CHATTING_ROOM"); //It does a task of joining the room

    socketClient.emit("JOINROOMSUCCESS", " Client joined room successfully");

    //server is emiting event JOINROOMSUCCESS to client in JOINROOMSUCCESS
  });


  //SENDMESSAGE TO THE ROOM Message
  socketClient.on("SENDROOMMESSAGE", (clientData) => {
    console.log(clientData, "klll");

    io.to("CHATTING_ROOM").emit("SENDROOMMESSAGE", clientData);
    //server is emiting event JOINROOMSUCCESS  to client
  });
});




app.get("/", (req, res) => {
  res.send("Hello World");
});

//backend
```

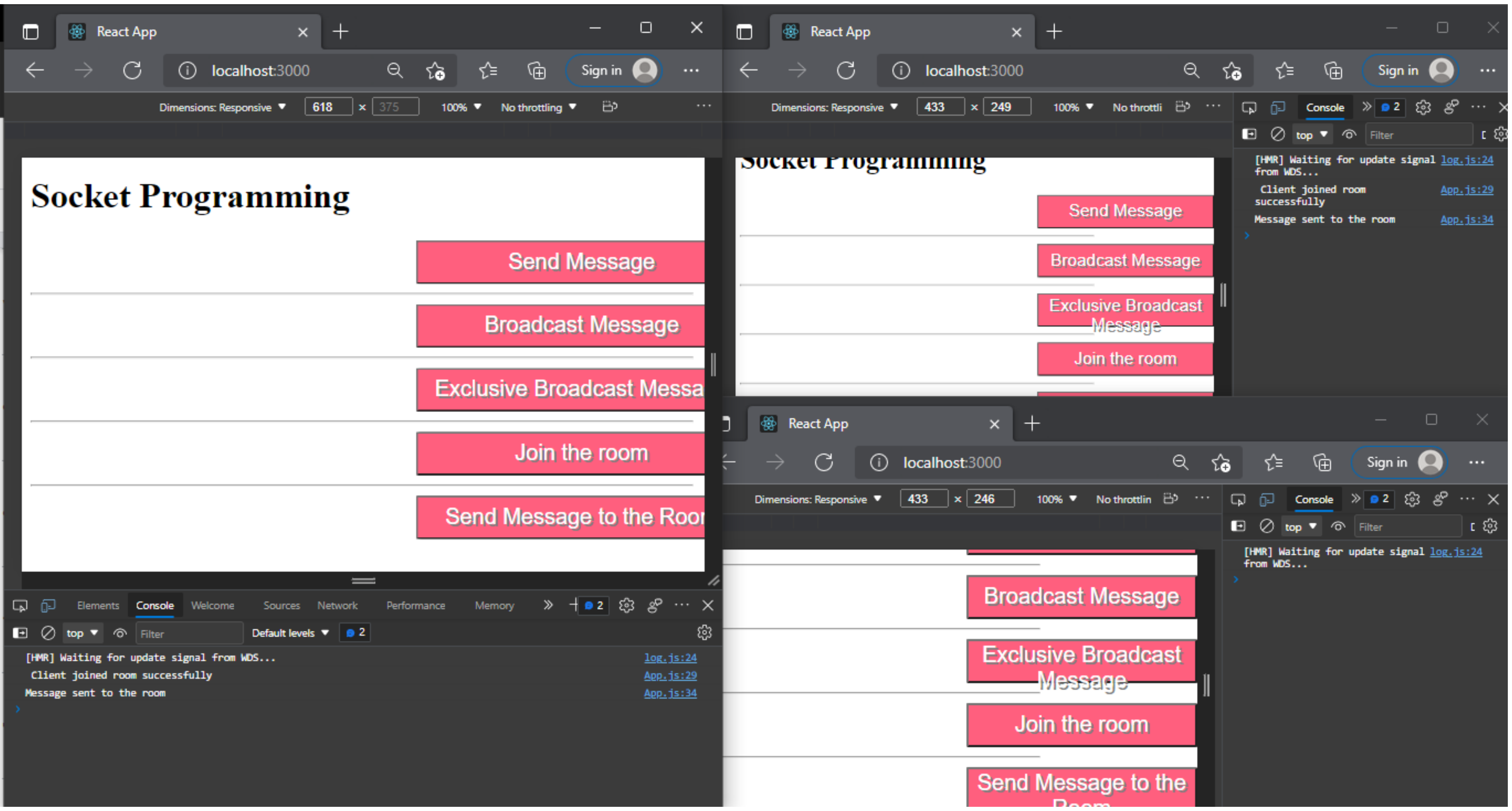
## Browser



```
[HMR] Waiting for update signal from WDS...        log.js:24
 Client joined room successfully                   App.js:29
```

```
[HMR] Waiting for update signal   log.js:24
from WDS...
 Client joined room               App.js:29
successfully
```

```
[HMR] Waiting for update signal   log.js:24
from WDS...
```



```
[HMR] Waiting for update signal from WDS...        log.js:24
 Client joined room successfully                   App.js:29
Message sent to the room                           App.js:34
```

```
[HMR] Waiting for update signal   log.js:24
from WDS...
 Client joined room               App.js:29
successfully
Message sent to the room          App.js:34
```

```
[HMR] Waiting for update signal   log.js:24
from WDS...
```