Eze Sunday Eze
Posted on Mar 1, 2019

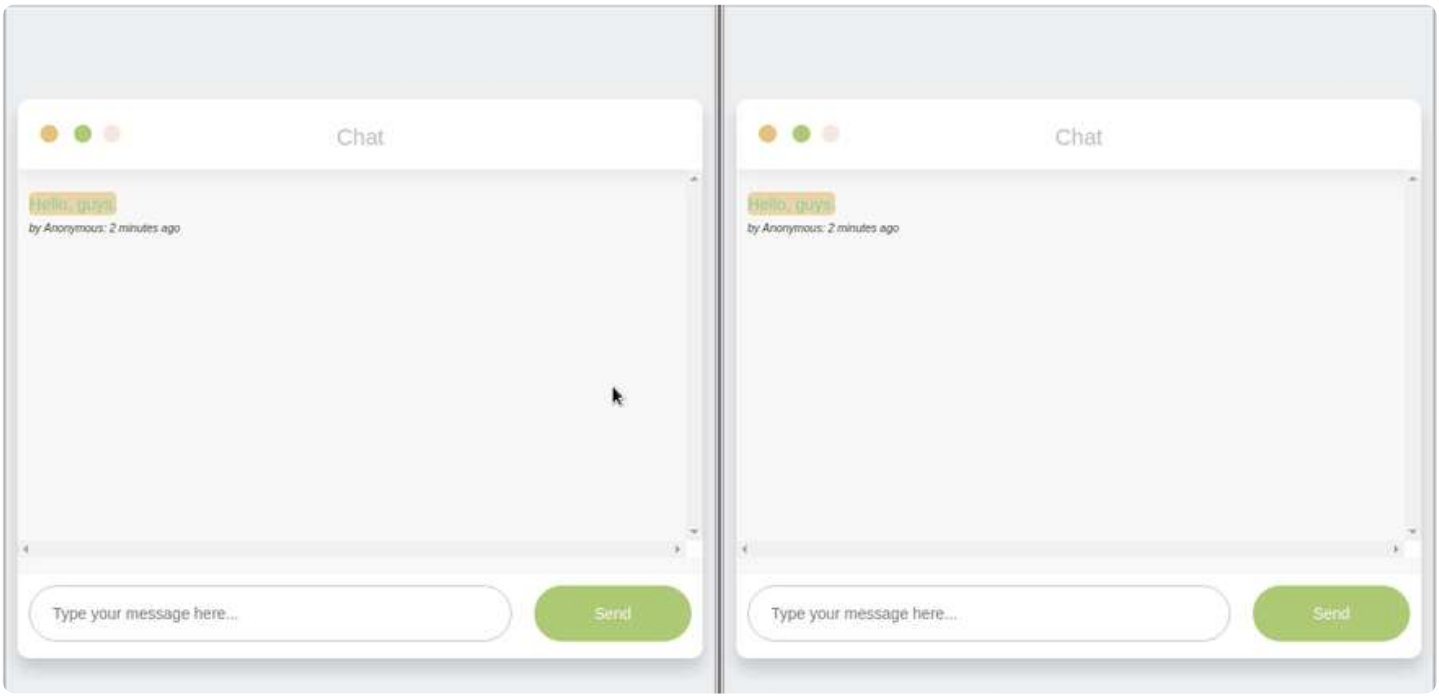
How to Build a Real-time Chat App With NodeJS, Socket.IO, and MongoDB

#socketio #node #mongodb #javascript

How to Build a Real-time Chat App With NodeJS, Socket.IO, and MongoDB

In this tutorial, we'll be building a real-time chat application with NodeJS, Express, Socket.io, and MongoDB.

Here is a screenshot of what we'll build:



Setup

I'll assume that you already have NodeJS and NPM installed. You can install it from the Node JS website if you don't have it installed already.

A basic Knowledge of Javascript is required.

Let’s get started.

Create a directory for the application, open the directory with your favourite editor such as Visual Studio Code. You can use any other editor, I’ll be using VS code in this tutorial:

```
mkdir chatApplication && cd chatApplication && code .
```

Next, let’s initialize the directory as a Nodejs application.

```
npm init
```

You’ll be prompted to fill in some information — that’s okay. The information will be used to set up your `package.json` file.

Dependencies Installation

Let's install our application's dependencies.

We'll be using the `express` web server to serve our static files and `body-parser` extract the entire body portion of an incoming request stream and exposes it to an API endpoint. So, let's install them. You'll see how they are used later in this tutorial.

```
npm install express body-parser --save
```

We added the --save flag so that it’ll be added as a dependency in our `package.json` file.

Note:

> Please, don’t use express generator as I won’t cover how to configure `socket.io` to work with express generator setup.

Next, install the mongoose node module. It is an ODM (Object Document Mapper) for MongoDB and it'll make our job a lot easier.

Let’s install it alongside socket.io and bluebird. Socket.IO is a JavaScript library for real-time web applications. **Bluebird** is a fully-featured Promise library for JavaScript.

```
npm install mongoose socket.io bluebird --save
```

That’s it for the Nodejs backend module installation.

Our `package.json` file should look like this now.

```
{
    "name": "chatApplication",
    "version": "1.0.0",
    "description": "",
    "main": "app.js",
    "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node app"
    },
    "author": "",
    "license": "ISC",
    "dependencies": {
        "bluebird": "^3.5.3",
        "body-parser": "^1.18.3",
        "express": "^4.16.4",
        "mongoose": "^5.4.14",
        "socket.io": "^2.2.0"
    }
}
```

Another way to install the above packages is to copy the `package.json` file above and paste it into your `package.json` file and run:

```
npm install
```

It'll install all the required packages.

Let’s set up the client side.

```
<!doctype  html>
<html>
    <head>
        <title>Anonymouse Real-time chat</title>
        <link  href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.0/css/bootstrap.min.cs
        <!------ Include the above in your HEAD tag ---------->
        <link  href="/css/style.css"  type="text/css"  rel="stylesheet"/>
    </head>
<body>
<div  class="chat_window">
    <div  class="top_menu">
    <div  class="buttons">
```

```
    <div  class="button close"></div>
    <div  class="button minimize"></div>
    <div  class="button maximize"></div>
</div>
    <div  class="title">Chat</div>
</div>
    <ul id="messages"  class="messages"></ul>
<div  class="bottom_wrapper clearfix">
<i  id="typing"></i>
    <form  id="form">
        <div  class="message_input_wrapper">
        <input  id="message"  class="message_input"  placeholder="Type your message here.
        </div>
        <button  class="send_message">Send</button>
    </form>
</div>
</div>
<script  src="/js/socket.js"></script>
<script  src="https://code.jquery.com/jquery-1.11.1.min.js"></script>
<script  src="https://cdn.jsdelivr.net/gh/rexeze/formatTimeStamp/src/index.min.js"></scri
<script  src="/js/chat.js"></script>
</body>
</html>
```

To connect Socket.IO server to the client we add the Socket.IO client-side javascript library.

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

That will be our html file for the frontend. You can grab the entire code for the frontend here to follow along. The best way to learn is to follow along.

You can download the client-side socket.io library here.

And here `/js/chat.js` is where we'll have our custom client-side javascript code.

Setting up our express server:

Create an `App.js`. You can call it server.js if you like.
It's my personal preference to call it App.js.

Inside the App.js file let's create and configure the express server to work with socket.io.

App.js

```
//Require the express moule
const express = require("express");

//create a new express application
const app = express()

//require the http module
const http = require("http").Server(app)

// require the socket.io module
const io = require("socket.io");

const port = 500;

const socket = io(http);
//create an event listener

//To listen to messages
socket.on("connection", (socket)=>{
console.log("user connected");
});

//wire up the server to listen to our port 500
http.listen(port, ()=>{
console.log("connected to port: "+ port)
});
```

This is the basic configuration required to set up socket.io in the backend.

Socket.IO works by adding event listeners to an instance of `http.Server` which is what we are doing here:

```
const socket = io(http);
```

Here is where we listen to new connection events:

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

For example, if a new user visits localhost:500 the message "user connected" will be printed on the console.

socket.on() takes an event name and a callback as parameters.

And there is also a special disconnect event that gets fire each time a user closes the tab.

```
socket.on(“connection”, (socket)=>{
    console.log(“user connected”);
    socket.on("disconnect", ()=>{
    console.log("Disconnected")
})
});
```

Setting up our frontend code

Open up your `js/chat.js` file and type the following code:

```
(function() {
    var  socket  =  io();
    $("form").submit(function(e) {
        e.preventDefault(); // prevents page reloading
        socket.emit("chat message", $("#m").val());
        $("#m").val("");
    return  true;
});
})();
```

This is a self-executing function it initializes socket.io on the client side and emits the message typed into the input box.

With this line of code, we create a global instance of the soicket.io client on the frontend.

```
 var  socket  =  io();
```

And inside the submit event handler, socket io is getting our chat from the text box and emitting it to the server.

```
$("form").submit(function(e) {
    e.preventDefault(); // prevents page reloading
    socket.emit("chat message", $("#m").val());
```

```
    $("#m").val("");
  return  true;
});
```

If you've gotten to this point, congratulations, you deserve some accolades.
😄

Great, we have both our express and socket.io server set up to work well. In fact, we've been able to send messages to the server by emitting the message from our input box.

```
socket.emit("chat message", $("#m").val());
```

Now from the server-side let's set up an event to listen to the "chat message" event and broadcast it to clients connected on port 500.

App.js

```
socket.on("chat message", function(msg) {
console.log("message: "  +  msg);
//broadcast message to everyone in port:5000 except yourself.
socket.broadcast.emit("received", { message: msg  });
});
});
```

This is the event handler that listens to the "chat message" event and the message received is in the parameter passed to the callback function.

```
socket.on("chat message", function(msg){
});
```

Inside this event, we can choose what we do with the message from the client ---insert it into the database, send it back to the client, etc.

In our case, we'll be saving it into the database and also sending it to the client.

We'll broadcast it. That means the server will send it to every other person connected to the server apart from the sender.

So, if Mr A sends the message to the server and the server broadcasts it, Mr B, C, D, etc will receive it but Mr A won't.

We don't want to receive a message we sent, do we?😭

That doesn't mean we can't receive a message we sent as well. If we remove the broadcast flag we'll also remove the message.

Here is how to broadcast an event:

```
socket.broadcast.emit("received",{message:msg})
```

With that out of the way, we can take the message received and append it to our UI.

If you run your application. You should see something similar to this. Please, don't laugh at my live chat. 💔

Wawu! Congratulations once again. let's add some database stuff and display our chats on the frontend.

Database Setup

Install MongoDB

Visit the mongoDB website to download it if you have not done so already.

And make sure your MongoDB server is running. They have an excellent documentation that details how to go about setting it up and to get it up and running. You can find the doc here.

Create Chat Schema

Create a file in the model's directory called `models/ChatSchema.js`
Nothing complex, we are just going to have 3 fields in our schema --- a message field, a sender field and a timestamp.

The `ChatSchema.js` file should look like this:

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const chatSchema = new Schema(
    {
    message: {
    type: String
    },
    sender: {
    type: String
        }
    },
        {
    timestamps: true
});

let Chat = mongoose.model("Chat", chatSchema);
module.exports = Chat;
```

Connection to the mongodb database

Create a file and name it `dbconnection.js`. That's where our database connection will live.

```
const mongoose = require("mongoose");
mongoose.Promise = require("bluebird");
const url = "mongodb://localhost:27017/chat";
const connect = mongoose.connect(url, { useNewUrlParser: true });
module.exports = connect;
```

Insert messages into the database

Since we are going to insert the messages in the server-side we'll be inserting the messages we receive from the frontend in the `App.js` file.

So, let's update the App.js file.

```
...
//database connection
const  Chat  = require("./models/Chat");
const  connect  = require("./dbconnect");



//setup event listener
socket.on("connection", socket  =>  {
    console.log("user connected");
    socket.on("disconnect", function() {
    console.log("user disconnected");
    });
    socket.on("chat message", function(msg) {
        console.log("message: "  +  msg);
        //broadcast message to everyone in port:5000 except yourself.
    socket.broadcast.emit("received", { message: msg  });

    //save chat to the database
    connect.then(db  =>  {
    console.log("connected correctly to the server");

    let  chatMessage  =  new Chat({ message: msg, sender: "Anonymous"});
    chatMessage.save();
    });
    });
});
```

We are creating a new document and saving it into the Chat collection in the database.

```
    let  chatMessage  =  new Chat({ message: msg, sender: "Anonymous"});
    chatMessage.save();
```

Display messages on the frontend

We'll, first of all, display our message history from the database and append all messages emitted by events.