Comprehensive Guide to Implementing Socket.IO in a Node.js MVC Pattern

Louis Trinh

Follow

androidstudio · December 17, 2024 (Updated: December 18, 2024) · Free: No

Create a project directory

```
Copy

mkdir socket-io-mvc-example cd socket-io-mvc-example
```

Initialize project:

```
Copy

npm init -y
```

Install dependencies:

```
Copy

npm install express socket.io
```

```
Copy

socket-io-mvc-example/
├── app.js          (Main server file)
├── controllers/
│   ├── index.js    (Express route controller)
│   └── sockets.js  (Socket.IO event handler)
├── models/
│   └── (Place your data models here)
├── views/
│   └── index.ejs   (Main view template)
└── package.json
```

app.js (Main Server File):

```
Copy

const express = require('express');
const app = express();
const http = require('http').Server(app);
const io = require('socket.io')(http);
// Load controllers (can be refactored into a separate loader function)
const indexController = require('./controllers/index');
const socketController = require('./controllers/sockets');
// Set view engine (adjust based on your preference)
app.set('view engine', 'ejs');
```

```
app.use('/', indexController);
// Socket.IO setup
socketController(io);
// Start server
http.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

controllers/index.js (Express Route Controller):

```
Copy

const express = require('express');
const router = express.Router();
router.get('/', (req, res) => {
  res.render('index'); // Render the main view
});
module.exports = router;
```

Socket.IO Event Handler (`controllers/sockets.js`):

This file defines the event handling logic for Socket.IO interactions.

```
Copy
```

```
    console.log('A user connected');

    // Handle chat messages sent from the client
    socket.on('chat message', (msg) => {
      // Broadcast the message to all connected clients
      io.emit('chat message', msg);

      // Alternatively, handle message logic here (e.g., saving to database)
      // You might interact with your data models defined in the `models` directory.
    });

    // Handle socket disconnection
    socket.on('disconnect', () => {
      console.log('A user disconnected');
    });
  });
};
```

Data Models (`models` Directory):

This directory can hold your application's data models, depending on your project's requirements. If you need to persist chat messages or user data, create model files (e.g., `message.js` or `user.js`) to represent your data structures and define functions for interacting with your database.

```
Copy

const mongoose = require('mongoose');
const MessageSchema = new mongoose.Schema({
  content: { type: String, required: true },
  sender: { type: String }, // Can be a user ID
  timestamp: { type: Date, default: Date.now },
});
module.exports = mongoose.model('Message', MessageSchema);
```

Main View Template (`views/index.ejs`):

This file defines the HTML structure and JavaScript code for the client-side interface.

```
Copy

<!DOCTYPE html>
<html>
<head>
  <title>Socket.IO Example</title>
  <script src="/socket.io/socket.io.js"></script>
  <script>
    const socket = io(); // Connect to Socket.IO server// Handle incoming chat message
    socket.on('chat message', (msg) => {
      const chatArea = document.getElementById('chat-area');
```

```
    function sendMessage() {
      const messageInput = document.getElementById('message-input');
      const message = messageInput.value;
      socket.emit('chat message', message);
      messageInput.value = ''; // Clear message input
    }
  </script>
</head>
<body>
  <h1>Chat Room</h1>
  <div id="chat-area"></div>
  <input type="text" id="message-input" placeholder="Enter your message">
  <button onclick="sendMessage()">Send</button>
</body>
</html>
```

Running the Application:

- From the project directory, run:

```
Copy

node app.js
```

- Open http://localhost:3000/ in your browser(s). You should see multiple browser windows/tabs connected to the same chat room.

- Implement user authentication and authorization to track users and potentially restrict access to chat functionalities.
- Integrate data models to persist chat messages in a database for historical viewing or retrieval.
- Expand on event handling to support additional functionalities like private messaging, user joining/leaving notifications, etc.
- Consider using a templating engine like EJS for more dynamic views.

#nodejs #express #socketio #mvc