4. Add the following `GET` route handlers:

Route	Description
`/`	Returns JSON for all paintings
`/:id`	Returns for just a single painting
`/gallery/:id`	Returns all paintings for a specific gallery id
`/artist/:id`	Returns all paintings for a specific artist id
`/year/min/max`	Returns all paintings whose `yearOfWork` field is between the two supplied values.

Guidance and Testing

1. Break this down into small steps and test after each step.

PROJECT 2:

DIFFICULTY LEVEL: Intermediate

Overview

In this project, you will be creating a full CRUD API.

Instructions

1. You have been provided a folder named **project2,** that contains the data and other files needed for this project. Use `npm init` to set up the folder, and `npm install` to add express.
2. Add a static file handler for resources in the **static** folder.
3. The data for the APIs is contained in a supplied json file. Create a provider module for this file.
4. Add the following `GET` route handlers:

Route	Description
`/`	Returns JSON for all companies
`/:id`	Returns for just a single company

5. Add `PUT`, `POST`, and `DELETE` route handlers, to handle updating an existing company, inserting a new company, and deleting an existing company.
6. Use the supplied form **tester.html** to verify your APIs work as expected.

Guidance and Testing

1. Break this down into small steps and test after each step.
2. While there is a provided form that you can use to test your APIs, it is often easier to test your APIs using a tool such as Postman and Insomnia. We recommend that you install one of these tools and try testing your API with it.

PROJECT 3:

DIFFICULTY LEVEL: Intermediate

Overview

In this project, you will create a more sophisticated chat application.

Instructions

1. You have been provided a **folder** named **project3,** that contains the data and other files needed for this project. Use `npm init` to set up the folder, and `npm install` to add express.
2. Examine **chat-adv-client-markup-only.html** in the browser. It illustrates the markup of the finished version. You will be working with **chat-adv-client.html** that doesn't have the extra markup. You will be writing code in **chat-adv-client.js** to programmatically generate the markup based on the reception of messages from the server.
3. Your server code will need to maintain a list of user objects. For each new user, you will need to save the name and an id number, which should be a random number between 1 and 70; this number will be used by the chat client to display a profile picture from **https://randomuser.me**.

   Your server code will also have to emit the updated user list to all clients whenever a new user is added. Because it is a random number, it's possible that two users could have the same profile picture. For simplicity sake, assume that each user name is unique. On the client side, when it receives a message from the server that there is a new user, it should display a message and then regenerate the list of users in the left side of chat using the passed user list data.
4. Your chat client has a Leave button. When the user clicks this button, it should send a message to the server that this user has left and then hide the chat window. The server should then remove the user from its list, and then emit a message to all clients of this action and provide an updated user list. The remaining clients should display a message and then regenerate the list of users in the left side of chat using the passed user list data.
5. The chat client has a textbox and a Send button. When the Send button is clicked, it should display the message directly in the chat window and then send the message to the server. The server, when it receives a new message, should broadcast it out to all the other clients (but not to the one that generated the message). The other clients should display the message content, the user that created it, and the current time.
6. The chat client can thus display four types of messages in the chat window: a user joined message, a user has left message, another user's chat message, and the current user's chat message. Three relevant CSS classes have been provided: `.message-received`, `.message-sent`, and `.message-user`. Your client code should set the appropriate class depending on which message has been received.

Guidance and Testing

7. Test by opening multiple windows with different user names. Sending messages and leaving should work appropriately and look as shown in Figure 13.15.

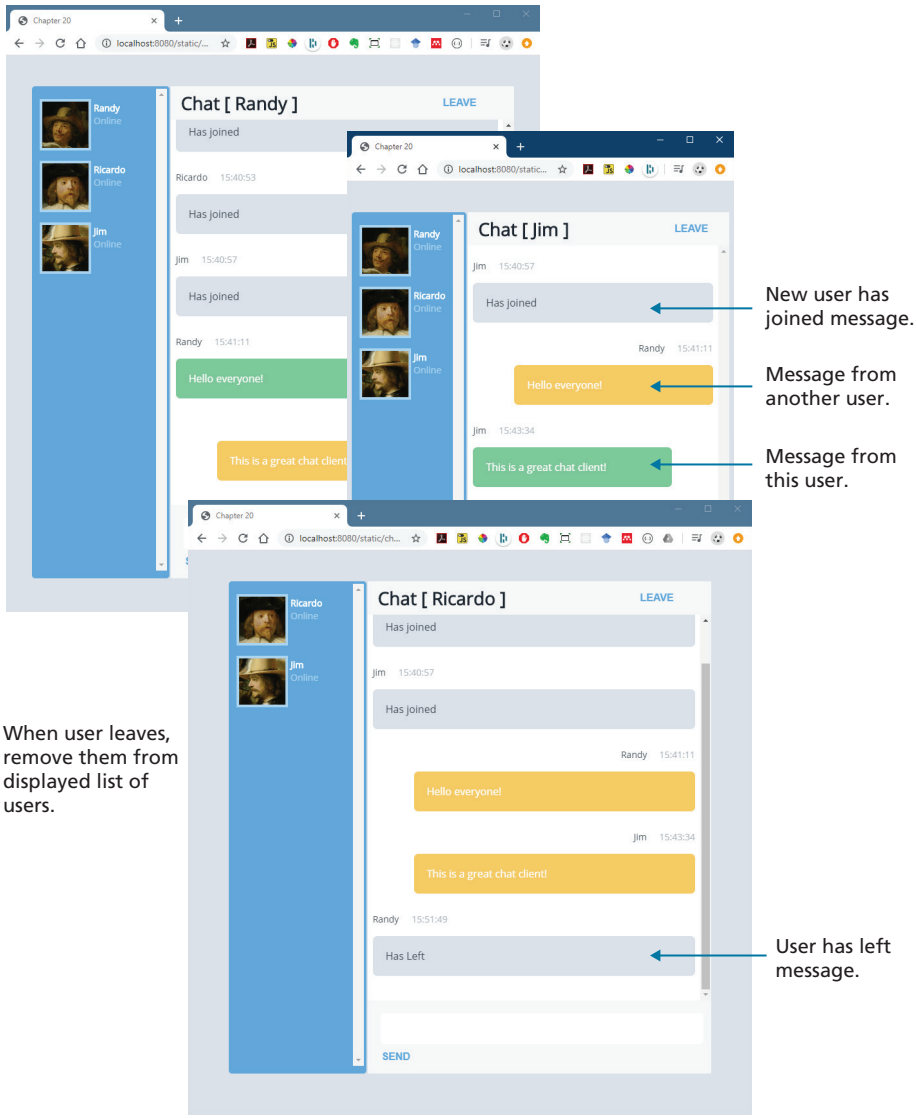

FIGURE 13.15 Completed Project #3

13.8.4 References

1. https://blog.risingstack.com/node-js-is-enterprise-ready/.
2. Slobodan Stojanović and Aleksandar Simović, *Serverless Applications with Node.js*, Manning Publications, 2018.
3. Jason Lengstorf, email correspondent.