**Challenge: A Chat Server**

**How long will this take?**

* 2 hours (Level 1)

**Overview: what is this challenge?**

In this challenge you must make an Express app which provides an API to manage a list of chat messages, in JSON format.

A simple front-end has been provided to allow you to test *some* of the functionality.

In the optional 'React' part of this challenge, you also make a React app which allows you to read, add and delete messages, backed by your new server.

**Pre-reqs**

*  You should have completed at least Level 1 of the Quote Server challenge before attempting this challenge.

**Want to run your code on the internet?**

If you want to share your server with other people the easiest way to do this is to use Glitch

*  Make sure you're logged in to <https://glitch.com/>
*  Remix <https://glitch.com/~cyf-chat-start>
*  Name your new server yourname-chat-server
*  Make sure you're logged in so that it saves
*  Check that it is working by making a request to /
*  Take time to read the comments
*  Copy the code you've written to Glitch

**Level 1 Challenge - make the chat server**

At this first level, your API must allow a client to:

*  Create a new message
*  Read all messages
*  Read one message specified by an ID
*  Delete a message, by ID
*  All message content should be passed as JSON.
*  Your routes should match the patterns established in class (RESTful). See the later spoiler section "Correct Routes" if you need the answer.

You can use [this chat tester client](https://cyf-chat-tester.netlify.com/) to test your routes.

**Data model**

Each chat message is an object with the following properties:

| **Name** | **Type** | **Example** |
| --- | --- | --- |
| id | number | 17 |
| from | string | "Neill" |
| text | string | "hi CYF! |

**Go ahead!**

If you think you know how to do that, go ahead!

Try to use what you know to do this challenge on your own. It does not require any new knowledge.

You may find useful the [express cheatsheet](https://github.com/nbogie/express-notes/blob/master/express-cheatsheet.md)

**End of Level 1 challenge!**

Well done!

What to do now:

*  *Don't* post on slack, unless there's a thread announced specifically for it.
*  Instead, attach the URLs as links when you "mark done" your assignment in Google Classroom.
*  You might want to download your project for safekeeping. (Tools: Git, Import, and Export: Download Project)

**Level 2 - simple validation**

For this level, your server must:

*  *reject* requests to create messages if the message objects have an empty or missing text or from property.
  +  In this case your server should return a status code of 400.

(Advanced note: people don't actually agree on the best status code for this situation.)

**A note on security**

There is intentionally no security or ownership of messages - anyone can delete one or all messages on your server.

This is a big topic for further study. We won't try to cover it in this challenge.

**Level 3 - more "read" functionality**

For this level your API *must* also allow a client to:

*  Read *only* messages whose text contains a given substring: /messages/search?text=express
*  Read only the most recent 10 messages: /messages/latest

**Level 4 - Optional - add a timestamp, timeSent**

For this level, the server must:

*  store a timestamp in each message object, in a field called timeSent.
*  This should be set to the current time when the server first receives the message. This should be a DateTime object, which can be created with new Date(). It will NOT be submitted by the client.

**Level 5 - Optional - add message *update* functionality**

If you want, you can also:

*  add support for the client to be able to *update* a message's text or from property. We'll cover this in the next week of the module, but you can research it easily.
*  Your server should NOT update the timeSent timestamp property during an update, if the client passes it back to you.

**Challenge: Advanced: Add a React app as a front-end**

Note: only do this if you have done all other Node homework this week - including Levels 1-3 of this challenge. The priority during the node module is *node*!

*  Make a very simple React app called chat-react-app

Note: Do not use create-react-app if you want your React code to be code-reviewed! Instead, you should:

*  fork our starting repo. See "How should I start my React app", below.

**Your UI should at least:**

*  Display the latest messages on load
*  Provide a "see latest" button to fetch and display the latest messages
*  Provide a "delete" button or a clickable icon next to each message.
  +  When clicked this should delete the message from the server and then from the local display.

**Optionally, your UI may also:**

*  Load and re-display the latest messages every 30 seconds.
*  Allow the user to use as much as possible of the back-end that you developed in levels 1-4 (e.g. message search).

**Don't forget:**

*  You'll have to enable CORS on the express app (see note below)
  + For how to post JSON, Read ["Using Fetch", on MDN](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch)
*  Host your react app on netlify
*  Attach the link in your google classroom submission (along with your glitch server url)

**How should I start my React app?**

To make code review easier for us, please:

*  fork our starting repo. (You can later create a PR when you want a code review.)

The repo is here: <https://github.com/CodeYourFuture/cyf-chat-react>

*  Follow the "Making a Pull Request" guide on <https://codeyourfuture.github.io/syllabus-master/others/making-a-pull-request.html>
*  Then run npm install after opening a terminal in the new project directory.
* Note that this repo was made by simply running create-react-app exactly as you have done in the past. There is nothing special about it!

**Example screenshot of Simple React app**

Here's an example of how your react app might look.

**Enabling CORS on the Express app**

You'll have to install and enable CORS on your server in order to allow your JSON to be loaded from a different server than your React app has been loaded from.

(Your react app has probably been loaded from the netlify.com domain and your server is at glitch.me).

On your express project on glitch, edit your package.json to add a dependency for the latest version of cors (e.g. "cors": "^2.8.5")

Then in your server.js add...

const cors = require('cors')

and

app.use(cors())

Read more or CORS [here](https://codeyourfuture.github.io/syllabus-master/others/cors.html) and in CORS in Express [here](https://expressjs.com/en/resources/middleware/cors.html).

**Spoiler: Correct Routes**

| **method** | **example path** | **behaviour** |
| --- | --- | --- |
| GET | /messages | return all messages |
| GET | /messages/17 | get one message by id |
| POST | /messages | create a new message |
| DELETE | /messages/17 | delete a message by id |