Using the Node.js Chrome Debugger

This guide will walk through the steps to use the Chrome Debugger with your server-side Node.js, giving you debugging features similar to the ones you've used in client-side JS. Current versions of Chrome have this feature built-in, so you can use it right away!

Starting the Debugger:

To start, add an --inspect flag to the usual nodemon command in your command line:

```
(base) mehovik@0-10-19-106-219 ~/work/cse154/wpl-queue $ nodemon --inspect app.js
[ nodemon ] 1.19.1
[ nodemon ] to restart at any time, enter 'rs'
[ nodemon ] watching: *.js
[ nodemon ] starting 'node --inspect app.js'
Debugger listening on ws://127.0.0.1:9229/05f5569e-c3ea-493b-a0a1-39f52ba6bd95
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
```

You can ignore the chrome-devtools:// URL that is displayed, but visit chrome://inspect in your Chrome browser. Then click “inspect” to start debugging your running Node.js program.

If you have a front-end view, you can alternatively now see a node icon in the browser Console:
Clicking “inspect” in the chrome://inspect view or the node icon in the localhost:8000/yourpage.html view will open the dedicated Node debugger, which you can set breakpoints in just like you would in your client-side JS.

Side-by-side view of client-side JS and Node.js Chrome debuggers. The “Enter Queue” click triggered a POST request to the /enqueue endpoint.

Just like in the client-side JS debugger, you can click the down arrow to step inside a function call. Below is an example inside the addToQueue called on line 107 of app.js, inspecting the result variable.

Below is a view of the RowDataPacket array after getWaitingCount is called on from line 109 of app.js (remember that SELECT queries return RowDataPacket arrays, INSERT queries return an OkPacket).
Seeing the results on the client-side!
Debugging/Testing Errors:
It’s not uncommon to forget to turn on your MAMP SQL server when using database-supported Node apps. The following screenshot catches the error when the db connection fails to connect due to MAMP not running (a good way to test your own 500 error-handling):

And on the client-side, we see:
Another common error is accessing an undefined db variable (`getDB()` does not return a db object when a connection error occurs).

You can also access your various variables in the Node.js debugger console, which can be pretty useful to test as well as explore different properties. Try it with the res, req, errors, SQL query results, etc.!