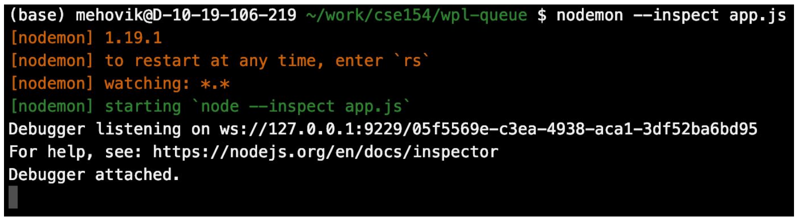
Using the Node.js Chrome Debugger

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!

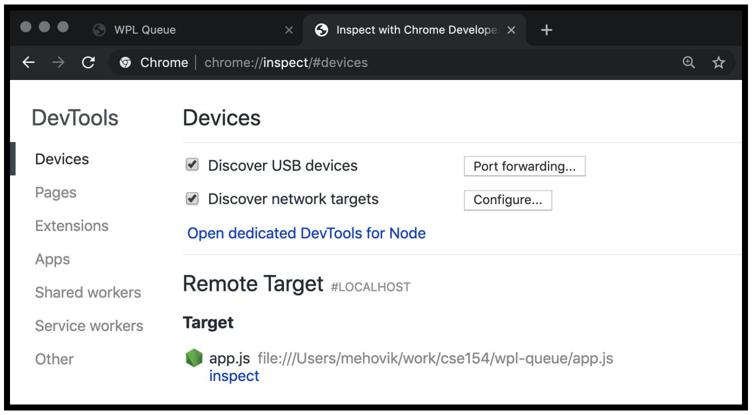
Starting the Debugger:

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



```
(base) mehovik@D-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: *.*
[nodemon] starting `node --inspect app.js`
Debugger listening on ws://127.0.0.1:9229/05f5569e-c3ea-4938-aca1-3df52ba6bd95
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.



Devices

- [✓] Discover USB devices — Port forwarding...
- [✓] Discover network targets — Configure...

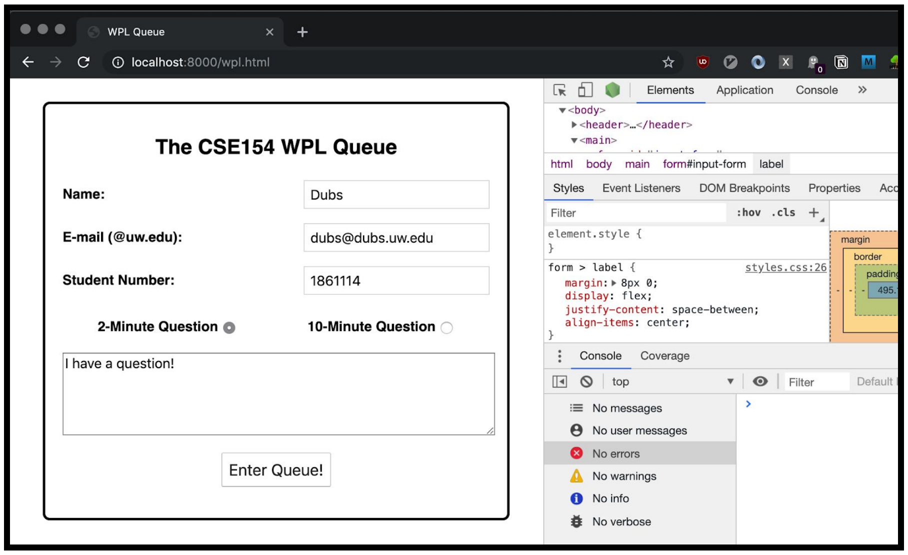
Open dedicated DevTools for Node

Remote Target #LOCALHOST

Target

app.js file:///Users/mehovik/work/cse154/wpl-queue/app.js
inspect

If you have a front-end view, you can alternatively now see a node icon in the browser Console:



The CSE154 WPL Queue

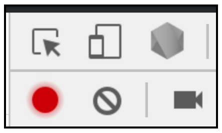
Name:	Dubs
E-mail (@uw.edu):	dubs@dubs.uw.edu
Student Number:	1861114

[✓ 2-Minute Question] 10-Minute Question

I have a question!

Enter Queue!

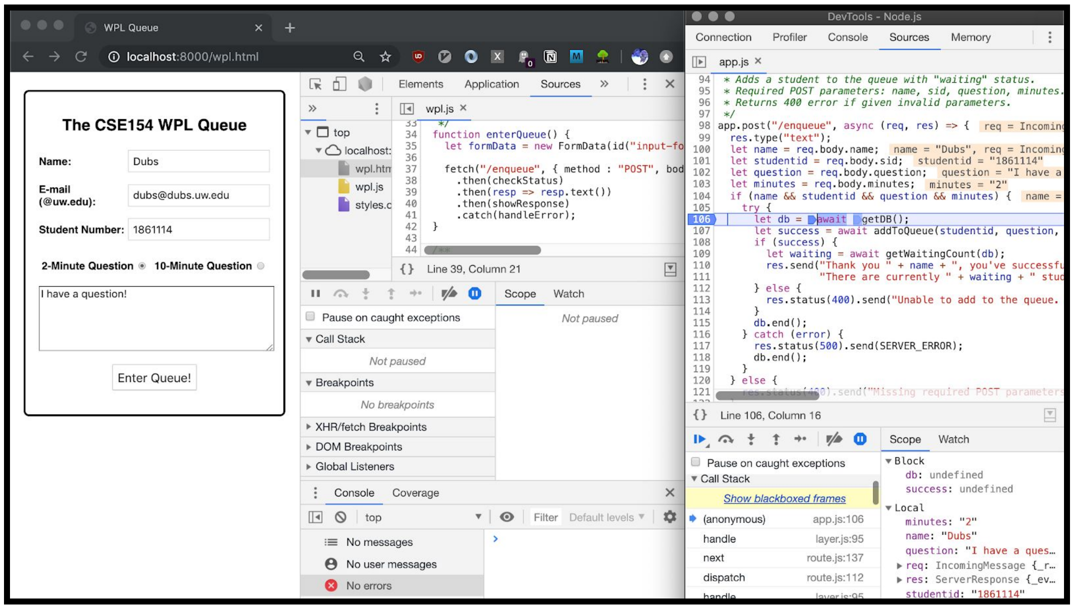
```
<body>
  <header>…</header>
  <main>

element.style {
}
form > label {                    styles.css:26
  margin: ▶ 8px 0;
  display: flex;
  justify-content: space-between;
  align-items: center;
}
```



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.



The CSE154 WPL Queue

Name: Dubs

E-mail (@uw.edu): dubs@dubs.uw.edu

Student Number: 1861114

2-Minute Question ● 10-Minute Question ○

I have a question!

```
  */
function enterQueue() {
  let formData = new FormData(id("input-fo

  fetch("/enqueue", { method : "POST", bod
    .then(checkStatus)
    .then(resp => resp.text())
    .then(showResponse)
    .catch(handleError);
}

/**
```

Line 39, Column 21

Pause on caught exceptions

Call Stack: Not paused

Breakpoints: No breakpoints

Scope: Not paused

No messages
No user messages
No errors

```
94   * Adds a student to the queue with "waiting" status.
95   * Required POST parameters: name, sid, question, minutes.
96   * Returns 400 error if given invalid parameters.
97   */
98  app.post("/enqueue", async (req, res) => {   req = Incoming
99    res.type("text");
100   let name = req.body.name;   name = "Dubs", req = Incoming
101   let studentid = req.body.sid;   studentid = "1861114"
102   let question = req.body.question;   question = "I have a
103   let minutes = req.body.minutes;   minutes = "2"
104   if (name && studentid && question && minutes) {   name =
105     try {
106       let db = await getDB();
107       let success = await addToQueue(studentid, question,
108       if (success) {
109         let waiting = await getWaitingCount(db);
110         res.send("Thank you " + name + ", you've successfu
111                  "There are currently " + waiting + " stud
112       } else {
113         res.status(400).send("Unable to add to the queue.
114       }
115       db.end();
116     } catch (error) {
117       res.status(500).send(SERVER_ERROR);
118       db.end();
119     }
120   } else {
121     res.status(400).send("Missing required POST parameters
```

Line 106, Column 16

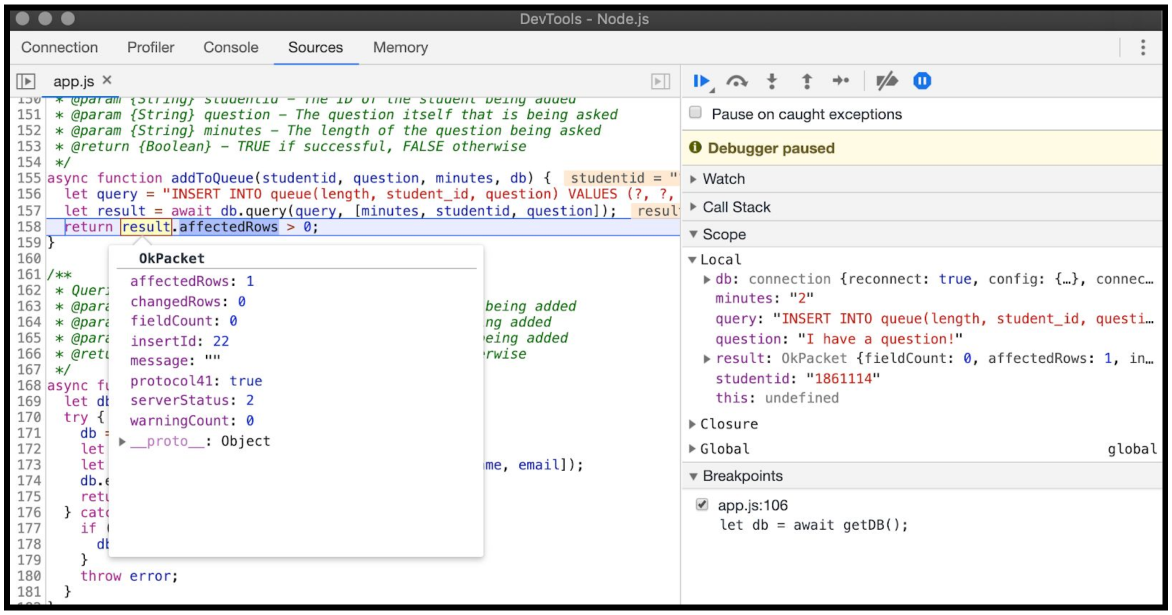
Pause on caught exceptions

Call Stack

```
(anonymous)    app.js:106
handle         layer.js:95
next           route.js:137
dispatch       route.js:112
handle         layer.js:95
```

```
▼ Block
    db: undefined
    success: undefined
▼ Local
    minutes: "2"
    name: "Dubs"
    question: "I have a ques…
  ▶ req: IncomingMessage {_r…
  ▶ res: ServerResponse {_ev…
    studentid: "1861114"
```

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.



```
150  * @param {String} studentid - The ID of the student being added
151  * @param {String} question - The question itself that is being asked
152  * @param {String} minutes - The length of the question being asked
153  * @return {Boolean} - TRUE if successful, FALSE otherwise
154  */
155 async function addToQueue(studentid, question, minutes, db) {   studentid = "
156   let query = "INSERT INTO queue(length, student_id, question) VALUES (?, ?,
157   let result = await db.query(query, [minutes, studentid, question]);   resul
158   return result.affectedRows > 0;
159 }
160
161 /**
162  * Quer
163  * @para                                         being added
164  * @para                                         ng added
165  * @para                                         eing added
166  * @retu                                         rwise
167  */
168 async fu
169   let db
170   try {
171     db =
172     let
173     let                                          me, email]);
174     db.e
175     retu
176   } catc
177     if (
178       db
179     }
180     throw error;
181   }
```

```
OkPacket
  affectedRows: 1
  changedRows: 0
  fieldCount: 0
  insertId: 22
  message: ""
  protocol41: true
  serverStatus: 2
  warningCount: 0
▶ __proto__: Object
```

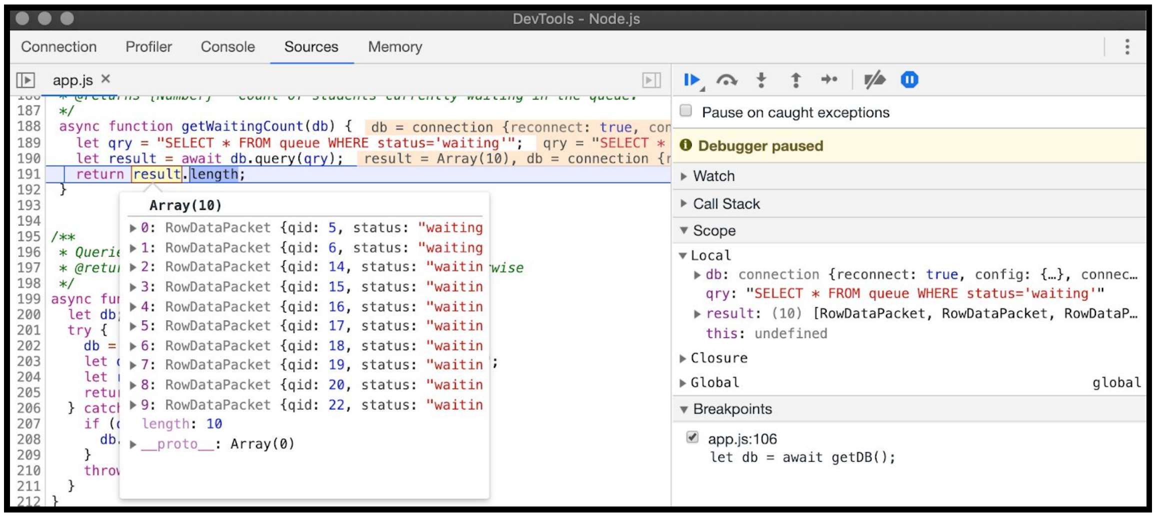
Pause on caught exceptions

Debugger paused

```
▼ Local
  ▶ db: connection {reconnect: true, config: {…}, connec…
    minutes: "2"
    query: "INSERT INTO queue(length, student_id, questi…
    question: "I have a question!"
  ▶ result: OkPacket {fieldCount: 0, affectedRows: 1, in…
    studentid: "1861114"
    this: undefined
▶ Closure
▶ Global                                          global
```

app.js:106
let db = await getDB();

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`).



```
187   */
188  async function getWaitingCount(db) {  db = connection {reconnect: true, con
189    let qry = "SELECT * FROM queue WHERE status='waiting'";  qry = "SELECT *
190    let result = await db.query(qry);  result = Array(10), db = connection {r
191    return result.length;
192  }
193
194
195  /**
196   * Querie
197   * @retu                                                 wise
198   */
199  async fu
200    let db;
201    try {
202      db =
203      let
204      let
205      retu
206    } catch
207      if (
208        db.
209      }
210      throw
211    }
212  }
```

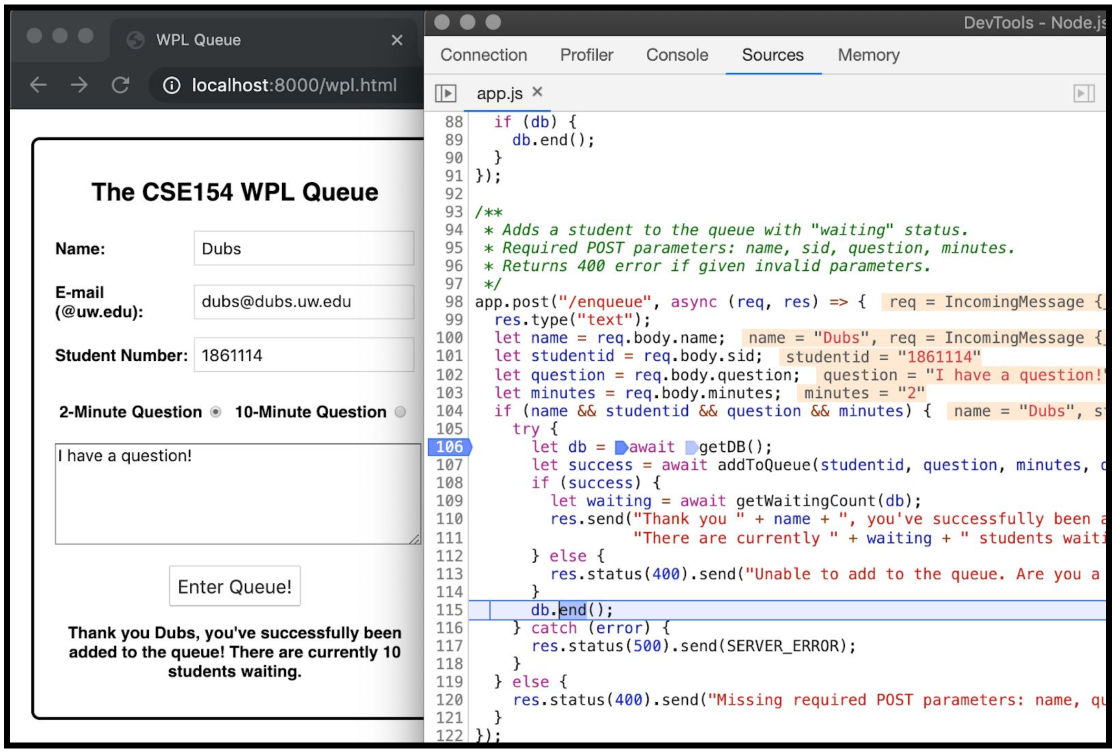
```
Array(10)
▶ 0: RowDataPacket {qid: 5, status: "waiting
▶ 1: RowDataPacket {qid: 6, status: "waiting
▶ 2: RowDataPacket {qid: 14, status: "waitin
▶ 3: RowDataPacket {qid: 15, status: "waitin
▶ 4: RowDataPacket {qid: 16, status: "waitin
▶ 5: RowDataPacket {qid: 17, status: "waitin
▶ 6: RowDataPacket {qid: 18, status: "waitin
▶ 7: RowDataPacket {qid: 19, status: "waitin
▶ 8: RowDataPacket {qid: 20, status: "waitin
▶ 9: RowDataPacket {qid: 22, status: "waitin
  length: 10
▶ __proto__: Array(0)
```

Pause on caught exceptions

Debugger paused

```
▶ Watch
▶ Call Stack
▼ Scope
▼ Local
  ▶ db: connection {reconnect: true, config: {…}, connec…
    qry: "SELECT * FROM queue WHERE status='waiting'"
  ▶ result: (10) [RowDataPacket, RowDataPacket, RowDataP…
    this: undefined
▶ Closure
▶ Global                                            global
▼ Breakpoints
  ✓ app.js:106
    let db = await getDB();
```

Seeing the results on the client-side!



The CSE154 WPL Queue

Name:	Dubs
E-mail (@uw.edu):	dubs@dubs.uw.edu
Student Number:	1861114

[✓ 2-Minute Question] 10-Minute Question

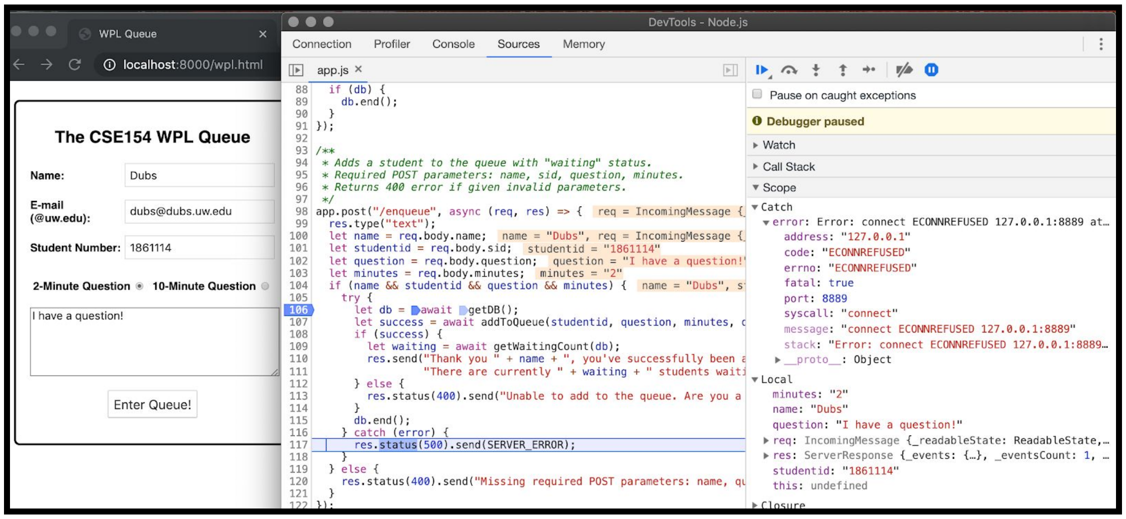
I have a question!

Thank you Dubs, you've successfully been added to the queue! There are currently 10 students waiting.

```
88     if (db) {
89       db.end();
90     }
91   });
92
93   /**
94    * Adds a student to the queue with "waiting" status.
95    * Required POST parameters: name, sid, question, minutes.
96    * Returns 400 error if given invalid parameters.
97    */
98   app.post("/enqueue", async (req, res) => {  req = IncomingMessage {
99     res.type("text");
100    let name = req.body.name;  name = "Dubs", req = IncomingMessage {
101    let studentid = req.body.sid;  studentid = "1861114"
102    let question = req.body.question;  question = "I have a question!"
103    let minutes = req.body.minutes;  minutes = "2"
104    if (name && studentid && question && minutes) {  name = "Dubs", s
105      try {
106        let db = await getDB();
107        let success = await addToQueue(studentid, question, minutes, d
108        if (success) {
109          let waiting = await getWaitingCount(db);
110          res.send("Thank you " + name + ", you've successfully been a
111                   "There are currently " + waiting + " students waiti
112        } else {
113          res.status(400).send("Unable to add to the queue. Are you a
114        }
115        db.end();
116      } catch (error) {
117        res.status(500).send(SERVER_ERROR);
118      }
119    } else {
120      res.status(400).send("Missing required POST parameters: name, qu
121    }
122  });
```

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):



WPL Queue — localhost:8000/wpl.html

The CSE154 WPL Queue

Name:	Dubs
E-mail (@uw.edu):	dubs@dubs.uw.edu
Student Number:	1861114

2-Minute Question ● 10-Minute Question ○

I have a question!

Enter Queue!

DevTools - Node.js — Connection, Profiler, Console, Sources, Memory — app.js

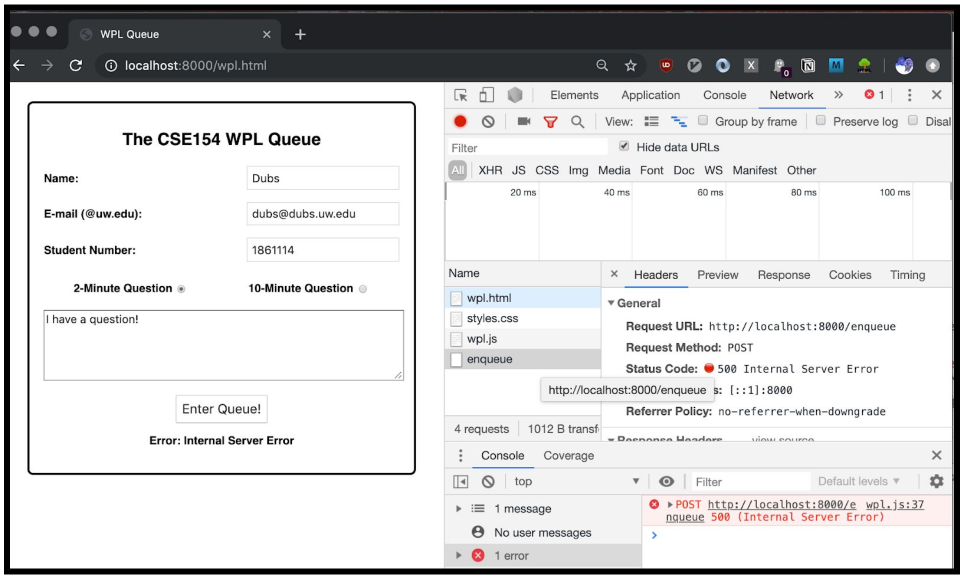
```
  if (db) {
    db.end();
  }
});

/**
 * Adds a student to the queue with "waiting" status.
 * Required POST parameters: name, sid, question, minutes.
 * Returns 400 error if given invalid parameters.
 */
app.post("/enqueue", async (req, res) => {   req = IncomingMessage {_
  res.type("text");
  let name = req.body.name;   name = "Dubs", req = IncomingMessage {_
  let studentid = req.body.sid;   studentid = "1861114"
  let question = req.body.question;   question = "I have a question!"
  let minutes = req.body.minutes;   minutes = "2"
  if (name && studentid && question && minutes) {   name = "Dubs", s
    try {
      let db = await getDB();
      let success = await addToQueue(studentid, question, minutes, c
      if (success) {
        let waiting = await getWaitingCount(db);
        res.send("Thank you " + name + ", you've successfully been a
                 "There are currently " + waiting + " students waiti
      } else {
        res.status(400).send("Unable to add to the queue. Are you a
      }
      db.end();
    } catch (error) {
      res.status(500).send(SERVER_ERROR);
    }
  } else {
    res.status(400).send("Missing required POST parameters: name, qu
  }
});
```

☐ Pause on caught exceptions

Debugger paused

```
▸ Watch
▸ Call Stack
▾ Scope
▾ Catch
  ▾ error: Error: connect ECONNREFUSED 127.0.0.1:8889 at…
      address: "127.0.0.1"
      code: "ECONNREFUSED"
      errno: "ECONNREFUSED"
      fatal: true
      port: 8889
      syscall: "connect"
      message: "connect ECONNREFUSED 127.0.0.1:8889"
      stack: "Error: connect ECONNREFUSED 127.0.0.1:8889…
    ▸ __proto__: Object
▾ Local
    minutes: "2"
    name: "Dubs"
    question: "I have a question!"
  ▸ req: IncomingMessage {_readableState: ReadableState,…
  ▸ res: ServerResponse {_events: {…}, _eventsCount: 1, …
    studentid: "1861114"
    this: undefined
▸ Closure
```

And on the client-side, we see:



WPL Queue — localhost:8000/wpl.html

The CSE154 WPL Queue

Name:	Dubs
E-mail (@uw.edu):	dubs@dubs.uw.edu
Student Number:	1861114

2-Minute Question ● 10-Minute Question ○

I have a question!

Enter Queue!

Error: Internal Server Error

Elements, Application, Console, Network — View: ☐ Group by frame ☐ Preserve log — Filter ☑ Hide data URLs — All XHR JS CSS Img Media Font Doc WS Manifest Other

Name
wpl.html
styles.css
wpl.js
enqueue

4 requests | 1012 B transf

Headers, Preview, Response, Cookies, Timing

▾ General

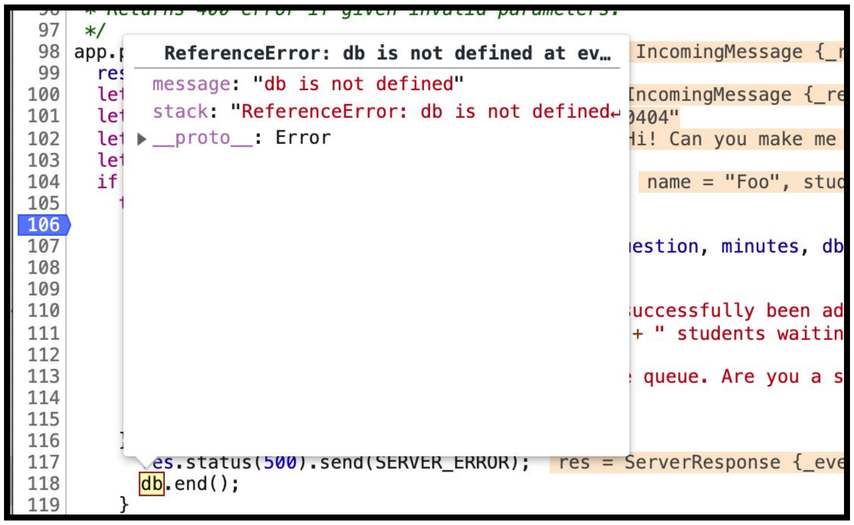
```
Request URL: http://localhost:8000/enqueue
Request Method: POST
Status Code: ● 500 Internal Server Error
http://localhost:8000/enqueue s: [::1]:8000
Referrer Policy: no-referrer-when-downgrade
```

Console — top — Default levels

```
1 message
No user messages
1 error
```

```
▸ POST http://localhost:8000/enqueue 500 (Internal Server Error)   wpl.js:37
```

Another common error is accessing an undefined db variable (`getDB()` does not return a db object when a connection error occurs).



```
ReferenceError: db is not defined at ev…
message: "db is not defined"
stack: "ReferenceError: db is not defined↵
▶ __proto__: Error
```

```
 */
app.
  res
  let
  let
  let
  let
  if
     res.status(500).send(SERVER_ERROR);   res = ServerResponse {_eve
    db.end();
  }
```

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.!