Server crashes on browser refresh.

For this assignment, you're tasked with completing the backend piece of a to-do application. The frontend client is already written and does two things:

1. When the applications loads, it sends a request to the backend to retrieve all to-do items for the client
2. When a new to-do item is submitted, the client sends an AJAX request to save the new item in the backend

Download Lab9.zip and extract the contents. The file index.html contains the base page for the client, which also loads in a JS file from the assets directory. The server.js file is the driver program for the backend application. This is the file you will be *running* from the Node.js CLI. The file todoList.js is a module that will contain all the code for handling the requests from the client. This is the file you will be *writing* your solution in.

Open up todoList.js and you'll notice a few things. First, the exported function, handleTodoList, has three parameters. The first two, req and res, are the request and response from the HTTP module. The third, session, is the object for the session belonging to the user making the request. This object can be used to store any data you wish, but it does come with one property, id, which contains the session id for the request (this will come in handy later). Next thing you'll notice is the switch statement in the handleTodoList function. Here we check the request method to appropriately handle incoming GET or POST requests. All other HTTP methods are not supported so we respond with a 405 error code

(Links to an external site.)

. Lastly, there is a helper function, convertRequest, at the bottom of the module. This function is there to properly convert any incoming POST requests into a JavaScript object. The function has two parameters. The first is the HTTP req object, and the second is a callback function which will be passed the converted object, data.

Part 1 (30 pts)

Let's start off by processing incoming POST requests. Before we do anything with the request, we want to maintain a list of to-do items for each session. Your handleTodoList function is already passed the session object, session, so we want to store our list in this object for later use. We will be creating a property called todoList, an array, on the session

object, but we only want to do this once, the first time that this user submits a to-do item. If there is already a property called todoList in this session, i.e. it's not undefined, then we will be pushing any new to-dos onto this list.

Now that we've verified our todoList in our session, we can call convertRequest to process the request. Remember, the data is **not** returned by the function, but rather passed into a **callback function**. Any work that you want to do with the data from the request should be done in the callback.

So what kind of work do we need to do with the data? For starters, each request from the client will look something like this:

```
{
  todo: "A new todo"
}
```

Into our todoList, we only want to push the string in the property todo, which you can access like this:

```
data.todo
```

Now that your session list has the new todo data, make sure to properly end the request by sending a response back like so in your callback function:

```
res.writeHead(200, 'OK');
res.end();
```

Part 2 (30 pts)

By now you may have noticed that when you refresh the page, all your to-dos disappear! Sorry, that doesn't mean you have nothing to do, it just means that we need to send back all the to-dos for each incoming GET request.

In the GET case of our switch statement, we will need to send back any to-dos in our session's todoList as JSON.

If the session's todoList hasn't been set (i.e. it's undefined) send back an empty list (array). This will ensure the client continues to work as expected if no to-dos have been added.

To send back the response, you will first have to format the todoList into the expected response format:

```
[
  {
    "id":0,
    "description":"study for quiz"
  },
  {
    "id":1,
    "description":"finish lab"
  }
]
```

Notice each item in the array is an object with two properties. The first property, id, can simply be the index of the to-do item in the todoList. The second property, description, is the string submitted in the POST request which we stored in our todoList.

Once you've converted the todoList into an array of objects with an id and a description, you must first convert them into a string before sending them in the response. You can do this with the following method:

```
JSON.stringify();
```

This method takes a JavaScript object and returns a string. you can then either use res.write() to add the string to the response or pass the string into res.end().

To finish the response, make sure you have the following lines:

```
res.writeHead(200, {'Content-Type': 'application/json'});
res.end();
```

You can optionally pass the JSON string to end, as mentioned before.

Part 3 (40 pts)

Your to-do list application might be looking good right about now. You can add to-do items and when you refresh the page, your list comes back. However, if you restarted your server at any point, you may have noticed that the list went blank. This isn't an accident. So far we have only been keeping our todo list *in memory* as an object. Any time that we

terminate our program, all that memory gets freed and we lose our list. To solve this problem, let's save our todo list to somewhere more persistent, like a file! Start off by importing the fs module using the require function.

```
const fs = require('fs');
```

Turn back to the POST portion of our handleTodoList function. After we push a new to-do into our todoList, let's save the updated list to a file using the fs.writeFile() function. The first argument of writeFile is the path of the file. This path has to be unique to the session we're going to be saving so let's use the session id as the file name. To keep our saved sessions organized we're going to be saving them in the sessions directory, so your full path name might look something like this:

```
./sessions/68bcd96055d6f3f89754f3b8b88b0476
```

The second argument of the writeFile function is the data. This data has to be a string so once again, use the JSON.stringify() function, this time to turn the entire session object into a string. The final argument to writeFile is a callback function. This is important because in Node.js, file I/O is an asynchronous operation, meaning that the next line of code following a file write won't be run after the file was actually written. Let's move the code to end the response into the callback to make sure we don't send the OK until the file is actually done writing. If you refer to our file system examples from class, you will see that the callback has one parameter, err, which will have a value if an error occurs. Let's make sure to respond correctly in the event that there is an error writing the file, like so:v

```
if (err) {
  res.writeHead(500, {'Content-Type': 'text/html'});
  return res.end("500 Internal Server Error");
}
```

Next, let's make sure we read the file when GETing the to-do list. The first thing we should do in our GET case now is to read the corresponding file for this session id. Use the function fs.readFile(), which takes two arguments. The first argument is the path to the file and the second is a callback function. This callback function takes two parameters. The first parameter, err, will have a value if there was an error reading the file, for instance, if the file was not found. The second parameter, data, will have the file's contents as a string. Before we can do anything with the data, we have to convert it back to an object using the method JSON.parse() which takes a string and returns an object. Remember, we wrote the entire session object to the file, therefore, we're only interested in the todoList property in the data we read from the file. Update the session's todoList property to be the todoList

from the file. Then, you will process this todoList like you did before to generate the GET response.