Open-Source Report

Proof of knowing your stuff in CSE312

Guidelines

Provided below is a template you must use to write your reports for your project.

Here are some things to note when working on your report, specifically about the **General Information & Licensing** section for each technology.

- **Code Repository**: Please link the code and not the documentation. If you'd like to refer to the documentation in the **Magic** section, you're more than welcome to, but we need to see the code you're referring to as well.
- **License Type**: Three letter acronym is fine.
- **License Description**: No need for the entire license here, just what separates it from the rest.
- **License Restrictions**: What can you *not* do as a result of using this technology in your project? Some licenses prevent you from using the project for commercial use, for example.

Also, feel free to extend the cell of any section if you feel you need more room.

If there's anything we can clarify, please don't hesitate to reach out! You can reach us using the methods outlined on the course website or see us during our office hours.

Express.js - TCP Connections

General Information & Licensing

Code Repository	https://github.com/expressjs/express
License Type	MIT License
License Description	https://github.com/expressjs/express/blob/master/LICENSE (The MIT License) Copyright (c) 2009-2014 TJ Holowaychuk <tj@vision-media.ca> Copyright (c) 2013-2014 Roman Shtylman <shtylman+expressjs@gmail.com> Copyright (c) 2014-2015 Douglas Christopher Wilson

	<doug@somethingdoug.com> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the 'Software'), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
License Restrictions	● No Warranty ● No Liability

Purpose

Express.js handles our TCP connection with the client. It is a web application framework for Node.js that abstracts the management of connections and protocols. The express application is exported from the express module and handles top-level functions.

In our use, it can create a TCP(Transmission Control Protocol) connection between the client and server. TCP, or Transmission Control Protocol, is a widely used protocol for transmitting data over a network. The purpose of this protocol is to provide an end-to-end, reliable, ordered delivery, connection-oriented network protocol when communicating between the client and server. This allows the server to communicate with the client and provide them with the information or services they have requested.

If there is no TCP connection, the server and the client will not be able to communicate with each other and exchange data. This means that the client will not be able to send requests to the server, and the server will not be able to send responses back to the client. As a result, the client will not be able to access any information or services provided by the server. This can cause the client's application or website to malfunction or not work at all.

This functionality was used in server.js where:

- https://github.com/Prakshal-Jain/NFTY/blob/main/backend/server.js
- Server instance (**server**) was imported from auction.js [Line 9]
    - Syntax → ***var{server,auction,app}=require('./routes/auction');***
- Used server.listen() to open TCP connections for the client [Line 41].
    - This will be used to set the port for our http server.

Background(auction.js):

- https://github.com/Prakshal-Jain/NFTY/blob/main/backend/routes/auction.js
- **http module** was imported in auction.js [Line 8]
- A server "object" is created using "***http.createServer(app)***" from Node.js [Line 12].
    - Creates a http server with no specified port.
- The **server** instance and the **app** are exported in an object using "***module.exports***" [Line 276].

Magic ⋆★˚｡⋆୨୧˚ ☽ ˚⁀➷｡˚★彡⋆༄

Background

The TCP connection is established by the listen method of the app object in the server.js file. This method is called with a **port number** as an argument, which specifies the port on which the server will listen for incoming requests. When the *listen* method is called, the server starts running and begins listening for requests on the specified port. This establishes the TCP connection, allowing the server to communicate with clients and handle their requests.

The **net module**

The express.js library uses the inbuilt functionalities of **node** module (dependency defined on Line-83) as the backbone and builds on top of it. For handling the TCP connection, the node library uses the NET module.
The **net module** in Node.js provides an asynchronous, stream-based API for working with Transmission Control Protocol (TCP) connections in Node.js. It can be used to create both **servers**, which listen for incoming connections from clients, and **clients**, which connect to a server to send and receive data.

The net module provides a number of different classes and methods for working with TCP connections. Here are the details of different parts of the **net module**, and how their implementation works:

- **createServer()** method: Defined on **Line-194**
    - The **createServer()** method in the net module is used to create a new net.Server object, which is a TCP server. When a client connects to the server, a new net.Socket object is created to represent the connection. The createServer() method allows you to provide a callback function that will be executed whenever a new client connects to the server. This callback function can be used to handle incoming data from the client and send data back to the client.
- **createConnection()** method: As explained in the comments of the library itself (Line-1286), creates a **new Socket** instance that can be used to connect to a server and send and receive data.
- An example of using net library is as follows:
    - We first create a new Server instance using the createServer() method. We then call the **listen()** method on the server instance, passing in the port number we want it to listen on. When the server successfully listens for incoming connections, the callback function provided to the listen() method will be called.
    - To create a new server instance, we use:
      const server = net.createServer();
    - To start listening for incoming connections on port 3000
      ```
      server.listen(3000, () => {
              console.log('server listening on port 3000');
      });
      ```

    - Once the Socket instance is connected to the server, we can use its methods to send and receive data over the connection. For example, we can use the **write()** method to send data to the server, and the **on()** method to listen for data events and handle incoming data from the server.

How the methods from net module relates to our **Homework 1**?

In **homework-1**, we implemented:

```
if __name__ == "__main__":
    host = "0.0.0.0"
    port = 8000
    server = socketserver.ThreadingTCPServer((host, port), MyTCPHandler)
    server.serve_forever()
```

In the implementation:

- **socketserver.ThreadingTCPServer** class is a subclass of the **socketserver.TCPServer** class that provides support for multiple threads handling requests concurrently. This means that when a client connects to the server and sends a request, a new thread is created to handle the request, allowing the server to continue accepting new connections and handling other requests simultaneously. It overrides the **socketserver.BaseRequestHandler** class that corresponds to the **createServer()** method under the hood from the net module in JavaScript (node) implementation.
- We call the **serve_forever()** method on the server instance to start it listening for incoming connections. This corresponds to the **listen()** method from the net module in JavaScript (node) implementation.

How the methods from net module relates to our group project implementation?

In our group project, we implemented:

```
const app = express();
const server = http.createServer(app);
.
.
.
server.listen(PORT, (error) => {...}
```

- **server.listen()** is a method used to bind and listen to connections on a specified port and host. In our implementation the port is **localhost:8000**. Since the server instance is created from the **app** application, it inherits all the methods of **app**.
    - **In the express library:** https://github.com/expressjs/express/blob/8368dc178af16b91b576c4c1d135f701a0007e5d/lib/application.js#L633
    - The function **app.listen()** is defined on lines **633-636**. As we can see, it calls **http.createServer(this)** under the hood.
    - General format of app.listen() → ***app.listen([port[, host[, backlog]]][, callback]).***

- In our implementation, we used:
    - Syntax → ***var server = http.createServer(this);***
        - **http.createServer()** is used because we wanted to reuse the HTTP server for **socket.io library** (websocket implementation for auctions).
        - **http.createServer((req, res) => { ... })** is a method in the **Node.js** HTTP module, and it is not defined in a specific Github repository. This method takes a callback function as an argument, which is called whenever the server receives a request. The ***callback function*** is passed into ***two arguments***: a request object that contains information about the request, and a response object that is used to send a response back to the client. The **Node.js** HTTP module is part of the core **Node.js** runtime, and it is not a separate package that is hosted on **Github**. You can learn more about the **Node.js** HTTP module and its methods, including **http.createServer**, in the official Node.js documentation.
        - Creates a **HTTP** server with the specified port. We used port 8000 therefore, the server is now bound to port 8000. [Line 634]
    - Syntax → ***return server.listen.apply(server, arguments);***
        - It then returns the server instance with the middleware for our example, we used function (err) which throws an error if the server can't start. [Line 635]