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.

Parsing HTTP headers

General Information & Licensing

Code Repository	https://github.com/ahssanja/CSE_312_Group_Project.git
License Type	Apache License 2.0
License Description	● permissive open-source license ● requires users to provide attribution to the original authors of the software ● includes a patent license, which grants users a license to any patents that are necessary to use the software
License Restrictions	● provides a limited warranty and liability protection for users, but it is not a guarantee of fitness for any particular purpose or non-infringement. ● does not include any copyleft provisions, which require users to share their changes or improvements with the community. ● does not grant users any rights to use the trademarks or logos associated with the software.

Magic ⋆★.ₒ˚.˚☽˚⁀➷｡˚★⋆｡˚ ༄

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does? Please explain this in detail, starting from after the TCP socket is created
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section will likely grow beyond the page

This report provides an overview of the code that parses HTTP headers in our tic tac toe app and the framework it uses. The frameworks our app is built on are Node.js and Express. It includes functionality for an authenticated user registration, login, leaderboard, and a Tic-Tac-Toe game with WebSocket communication.

Our code uses the Express framework to handle HTTP requests and responses and implement WebSockets. Once we import these modules the code proceeds to create an Express application using the express() function. From here our code defines routes using the app.get() and app.post() functions provided by Express.

For example

```
app.get('/register.html', (req, res) => {
    const fileName = 'register.html';
    const filePath = path.join(__dirname, 'public', fileName);
    res.sendFile(filePath);
});
```

Where once we get a request for GET /register.html we simply respond with the html page which directs to hour html page where users can register. The framework is able to take care of the headers here such as the 200 level response and the file length etc.

Another more complex example is

```
app.post('/login', async (req, res) => {
    try {
        const user = await User.findOne({ username: req.body.username });
        if (!user) {
            return res.status(400).send({ message: 'User not found' });
        }

        const validPassword = await bcrypt.compare(req.body.password, user.password);
        if (!validPassword) {
            return res.status(400).send({ message: 'Invalid password' });
        }

        const token = crypto.randomBytes(32).toString('hex');
        const hashedToken = await bcrypt.hash(token, 10);

        //user.hashedToken = hashedToken;
        //await user.save();

        res.render('LandingPage', { username: user.username });

    } catch (error) {
        res.status(500).send(error);
    }
});
```

Here once we get a request (once a user has asked to login) we first check the user's details and we check if those details match our database and only then do we render the new landing page html with the rendered content replacing the template with the client's username. Here the framework is very helpful as it changes the templates for us while we simply provide it with the username we want to replace it with.

All together these routes serve as endpoints for handling different types of HTTP requests from clients. The routes allow the application to handle other requests for our game's pages like index.html, register.html, LandingPage.html, and leaderboard.html, as well as static resources such as client.js. To parse the HTTP headers, the code utilizes the body-parser middleware. This middleware is responsible for parsing the request body, which can contain either URL-encoded form data or JSON payloads. The *bodyParser.urlencoded()* middleware is specifically used to parse URL-encoded bodies, while *bodyParser.json()* middleware is used for parsing JSON bodies.

In conclusion, our application employs various routes to handle different types of HTTP requests from clients, including requests for pages both dynamic for the game's purposes as well as static resources like client.js. To parse the HTTP headers and extract relevant information from the request bodies, the application uses the body-parser middleware. This middleware is capable of parsing both URL-encoded form data and JSON payloads, utilizing the bodyParser.urlencoded() and bodyParser.json() middleware, respectively.