JWT lab

1 Instructions

1. During this exercise, document your progress using screenshots (of the entire screen) to demonstrate that you have indeed completed all the required tasks.
2. Save all the screenshots in the GitHub project, in a dedicated folder called "proof", and present them chronologically in the README file. See the forum to understand what present in README means.
3. There's no need to make hundreds of screenshots, but please take enough to convince the person checking your submission that you completed the exercise as instructed.
4. Your README file must also include a link to the repository.
5. When you are ready to submit, download the repository from GitHub (the entire repository) and submit it. Your submission MUST NOT contain links to other locations where the files are (like Google Drive or a link to the GitHub repository instead of the actual files). You are allowed to submit a zip file.
6. The Moodle has a hard limit on the upload size, and will not allow you to upload files beyond 5MB. There is nothing I can do about it.

2 Json Web Token

In the past, we implemented a login form. In this targilon, we will implement a login authentication, using Json Web Token (JWT).

The steps are:

1. When a user submits a correct username and password, we are going to generate a special string for them, called a **token**. This token is basically a Json object, but, we use **cryptography** to ensure that this object is verifiable. Namely, only we (the server) can generate it - and the server can always **verify** that a token was indeed generated by the server and not someone else. The token will contain the username that was submitted.

2. The token will be sent back to the client.
3. The client will attach this token to every future request it sends to the server.
4. The server will *validate* the token, and will use it to understand who is the user, because the token will contain the username. No one has this token (except for the actual user), and no one else can forge it. So, it must be the real user that perform the request to the server!

You need to perform the following tasks:

- Create an empty NodeJS project.
- Create an app.js file with the following content:

```
// Load express
const express = require('express')
const app = express()

// Use a middleware that processes JSON objects in the request
app.use(express.json());
// Serve static files from the public folder
app.use(express.static('public'))
// Use a library to perform the cryptographic operations
const jwt = require("jsonwebtoken")

// We are using cryptography, to ensure that no one else will be able to impersonate users
// To that end, we are going to use the following secret key
// Keep it safe. No one except the server should know this secret value
const key = "Some super secret key shhhhhhhhhhhhhhhh!!!!!"

// Define a function that responds with a json response.
// Only logged in users should be able to execute this function
const index = (req, res) => {
    res.json({ data: 'secret data' })
}

// Ensure that the user sent a valid token
const isLoggedIn = (req, res, next) => {
    // If the request has an authorization header
    if (req.headers.authorization) {
        // Extract the token from that header
        const token = req.headers.authorization.split(" ")[1];

        try {
            // Verify the token is valid
            const data = jwt.verify(token, key);
            console.log('The logged in user is: ' + data.username);
            // Token validation was successful. Continue to the actual function (index)
            return next()
        } catch (err) {
            return res.status(401).send("Invalid Token");
        }
    }
    else
        return res.status(403).send('Token required');
}

// Handle login form submission
const processLogin = (req, res) => {

    // Check credentials
    if (req.body.username == 'guest' &&
```

```
        req.body.password == '123456') {

        // Correct username and password - Yayyyy

        // We now want to generate the JWT.
        // The token can contain whatever information we desire.
        // However, do not put sensitive information there, like passwords.
        // Here, we will only put the *validated* username
        const data = { username: req.body.username }

        // Generate the token.
        const token = jwt.sign(data, key)

        // Return the token to the browser
        res.status(201).json({ token });
    }
    else
        // Incorrect username/password. The user should try again.
        res.status(404).send('Invalid username and/or password')
}

// Handle login attempt
app.post('/login', processLogin)

// Show sensitive route index - only if logged in
app.get('/', isLoggedIn, index)

// Start server
app.listen(89)
```

- Create a 'public' folder with a file called login.html with the following content:

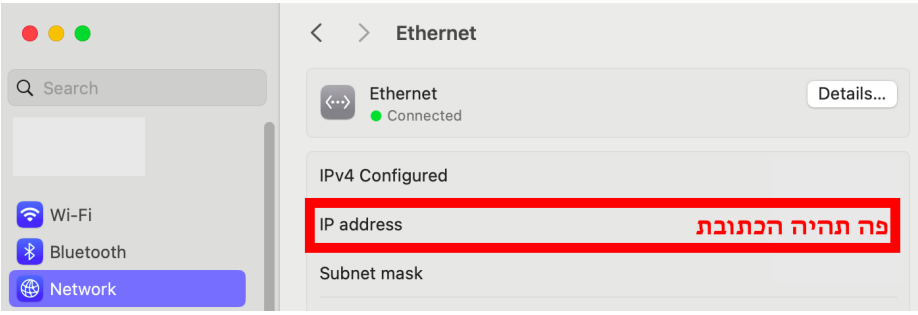
```
<html>

<body>
    <form method="post" action="http://localhost:89/login">
        <input name="username"></input>
        <input type="password" name="password"></input>
        <input type="submit"></input>
    </form>
    <script>
        document.forms[0].onsubmit = async (e) => {
            // Do not submit data synchrounously
            e.preventDefault();

            // Create a json object with the username and password from the form
            const data = {
                username: document.getElementsByTagName('input')[0].value,
                password: document.getElementsByTagName('input')[1].value
            }

            // Send post request to the server asynchrounously
            // fetch sends the asynchrounous request
            // The request is sent to the server according to the url that was set in: document.forms[0].action
            // The await keyword ensures that 'res' will have the result from the server.
            // even though 'fetch' is asynchrounous
            const res = await fetch(document.forms[0].action, {
                'method': 'post', // send a post request
                'headers': {
                    'Content-Type': 'application/json', // the data (username/password) is in the form of a JSON object
                },
                'body': JSON.stringify(data) // The actual data (username/password)
            }
            )
```

```
            // The server's response is a json object
            const json = await res.json()

            if (res.status != 201)
                alert('Invalid username and/or password')
            else {
                // Correct username/password
                // Take the token the server sent us
                // and make *another* request to the homepage
                // but attach the token to the request
                const res = await fetch('http://localhost:89/', {
                    'headers': {
                        'Content-Type': 'application/json',
                        'authorization': 'bearer ' + json.token // attach the token
                    },
                }
                )

                // Show the server's response
                const result = await res.json()
                alert('the secret data is: ' + result.data)
            }
        }
    </script>
</body>

</html>
```

- Install the relevant libraries, execute the server and demonstrate that:
  - You cannot access the homepage without performing login
  - Once you login, you can access the secret data from the homepage

3 Client-server

1. For this step, you will need two devices: a computer and a smartphone.
2. Make sure the two devices are on the same network. This can be the same home WIFI network or you can use your phone's tethering option (when the phone generates a wifi network, and connect you computer to that network). Either is fine.
3. Most likely it will **NOT** work at the university's wifi.
4. Once both computers are on the same wifi, get the ip address of the **computer**. Make sure the computer still runs the server from the previous step.
5. How to get the ip address? that depends on your operating system. You can use a terminal or the GUI of the OS. For example, if you have a Mac, you can go to settings, and see it like this: (in the picture - I'm on Network (left menu), but you will see it in 'Wi-Fi' because you are using wifi to connect to the network)



6. Other operating systems have similar screens in the 'network' settings.
7. Then, go to the phone, open a browser **on the phone**, and go to the same URL from the previous step, **but using the ip address**.
8. For example, you accessed the server in the previous step by going to: http://localhost:89/login.html
9. Now, on your phone, you cannot use 'localhost', **the server is not running on your phone, but on your computer**.
10. So, instead, in your phone, you need to go to: http://[ip address of computer]:89/login.html
11. Make sure that it works, and that you are indeed able to see the login page on your phone, served by the server that run's on your computer.