

Week 8: React and friends

Exercise Instructions:

During this exercise, document your progress using screenshots of the entire screen to demonstrate that you have completed all the required tasks. Save all screenshots in a folder named “proof” within your GitHub project, and organize them chronologically (e.g., 1.png, 2.png, etc.). Screenshots can be in any standard image format (e.g., png, jpeg).

You don't need hundreds of screenshots, but please include enough to clearly show completion of the required tasks. Your README file must also include a link to the repository.

For Submission:

Download the entire repository from GitHub and upload it to Moodle. Do not provide links to external locations (e.g., Google Drive or GitHub links). Note: Moodle has a 5MB upload limit. If necessary, remove screenshots from the submission to fit the size limit, but ensure that all screenshots remain in the GitHub repository.

Web

1. In Visual Studio code, copy the code from the following slides (file called in Moodle: 9 - UI), save it to html/css files and load it in a browser (separately):
 - a. Slide 35 (the code on the right - the HTML code)
 - b. Slide 47
 - c. Slide 56
 - d. Slide 59
 - e. Slide 66
 - f. Slide 68
2. Do (at least) five (total) **different** changes to the code of slide 68, both in the HTML and CSS.

React:

1. Create a new React project and push it to a new **private** GitHub repository with a README file. Remember, put node_modules in .gitignore.
2. Make sure you can run the application using npm start and see it in the browser.
3. Change the text in the code that came with the project, and observe the application in the browser automatically reflects the change.
4. Execute the code from the file in Moodle called: 9 - UI, slides number 147-149.
Make a few changes along the way, for example: changing text, tags etc.

Serve React App from Node

When you run your React app (e.g., by pressing F5), the application is automatically launched in your default browser. Behind the scenes, your code is served using a **web server**. Let's see how we can do it ourselves:

1. Create the default React app.
2. Instead of running the app using `npm start`, create a build version using `npm run build`.
3. This creates a directory called `build`, which contains the code after the build command was executed, ready to be served by a web server. Observe the files in this directory and compare them to the contents of the project folder.
4. Create a simple Node web server to serve static files from a `public` folder. (Hint: You've done this already in the past.)
5. Move the contents of the `build` folder into the `public` folder and run the server. Demonstrate and observe that the app is now served from *your* web server, **not** the default web server.

Json Web Token:

We will implement login authentication using JSON Web Token (JWT).

Steps:

1. When a user submits the correct username and password, generate a special string called a **token**. This token is a JSON object, but using **cryptography** we can make it verifiable. That is, only the server can generate and verify it.
2. The token contains the username and is sent back to the client.
3. The client attaches the token to every future request.
4. The server validates the token and uses it to identify the user. The JWT ensures that users can impersonate other users.

Tasks:

- Create a new NodeJS project.
- Create the following code files and **make sure you understand them**. If needed, use GenAI tools and ask them to explain to you lines of code you did not understand. Especially, make sure you understand the **code in red**
- Create an **app.js** file with the following content:

```
const express = require('express')
const app = express()
app.use(express.json());
app.use(express.static('public'))
const jwt = require("jsonwebtoken")
const key = "Some super secret key shhhhhhhhhhhhhhhhh!!!!"

const index = (req, res) => {
  res.json({ data: 'secret data' })
}

const isLoggedIn = (req, res, next) => {
  if (req.headers.authorization) {
    const token = req.headers.authorization.split(" ")[1];
    try {
```

```

    const data = jwt.verify(token, key);
    console.log('The logged in user is: ' + data.username);
    return next()
  } catch (err) {
    return res.status(401).send("Invalid Token");
  }
}
else
  return res.status(403).send('Token required');
}

```

```

const processLogin = (req, res) => {
  if (req.body.username == 'guest' &&
    req.body.password == '123456') {
    const data = { username: req.body.username }
    const token = jwt.sign(data, key)
    res.status(201).json({ token });
  }
  else
    res.status(404).send('Invalid username and/or password')
}

```

```

app.post('/login', processLogin)
app.get('/', isLoggedIn, index)
app.listen(89)

```

- Create a `public` folder with `login.html`:

```

<html>
<body>

```

```

<form method="post" action="http://localhost:89/login">
  <input name="username">
  <input type="password" name="password">
  <input type="submit">
</form>
<script>
  document.forms[0].onsubmit = async (e) => {
    e.preventDefault();
    const data = {
      username: document.getElementsByTagName('input')[0].value,
      password: document.getElementsByTagName('input')[1].value
    }
    const res = await fetch(document.forms[0].action, {
      'method': 'post',
      'headers': {
        'Content-Type': 'application/json',
      },
      'body': JSON.stringify(data)
    })
    const json = await res.json()
    if (res.status !== 201)
      alert('Invalid username and/or password')
    else {
      const res = await fetch('http://localhost:89/', {
        'headers': {
          'Content-Type': 'application/json',
          'authorization': 'bearer ' + json.token
        },
      },
    })
    const result = await res.json()
  }

```

```
        alert('the secret data is: ' + result.data)
    }
}
</script>
</body>
</html>
```

- Install the relevant libraries and test:
 - Cannot access homepage without login.
 - After login, the secret data is accessible.

End of week 10 mini-exercise