Exercises:

# Cookies and Storage

**1.1** Write a script that stores your name and hometown in localStorage.

**1.2a** Build a website that shows how many times the user has visited your website.

**1.2b** Add a text form where a user can type something. Below that, there should be a heading "Past Entries" that shows what the user has entered in the field before.

**1.2c** Make sure that "Past Entries" do not disappear if you reload the page.

**1.2d** Add a timestamp to each entry.

**1.2e** Add a button that resets the number of visits to zero.

**1.3a** Make a website that contains a button. If you press the button, the browser must store a cookie with the name clicks. clicks increases by 1 for each time the user presses the button.

**1.3b** Add a header to the page. This should say "Welcome back, you were here X seconds ago!"

**1.3c** Add a button "Remove all cookies" which removes all cookies on your page.

**1.4** Build a website that prints all cookies and their values in a table with two columns. For example:

| Name | Value |
| --- | --- |
| lastVisited | 2022-01-01T23:00:00.000Z |
| favoriteTheme | darkmode |
| itemsInCart | 2 |

# Sessions

**2** In this exercise, you will build a page for an online bookstore. Below are some features you can implement in any order. You do not need to implement any payment system.

* Users can see different books on the home page available in the bookstore. *Start with this one.*
* When you click on a book, you get some information about it. You can start with just a few books and hardcoded values. If you want, you can put these in a database. *Wait to enter the values into a database. Start with hardcoded values.*
* A user gets "Recently Viewed" on the home page with a list of books they have just looked at. *Use express session and put recently viewed in an array.*
* A user can add books to the cart. Which books are in the shopping cart are saved in the session. *Also use express session here*
* A user has a login where session data is saved more permanently. The user can now close the browser, come back and still have the goods in the basket. *First here you need redis.*
* After a certain amount of inactivity, the session is destroyed and the user is logged out. *You must add an expires to the cookie on your session, and continuously add time every time you send a request to the server.*

*Tip*

1. *Start by making an index.html with hardcoded books. Then make sure to send it from an express server.*
2. *Create an endpoint that returns the books*
3. *Download the books via index.html (you can import a script here)*
4. *Print the books via a .forEach()*
5. *Add buttons below each book in your .forEach() that lead to a POST /books*

Feel free to refactor the code as you learn better ways to solve the problems during the course. For example:

* Store data with Redis.
* Use the helmet.

# Radish

**3a** Insert a value with the favoriteColor key via the command prompt (CLI). Then get this value

**3b** Do the same as in 3a, but via Node.

# Hash and Salt

**4** Make a website that has a form with username and password. The form sends a POST request to /login. Also add a page to the address /protected.

1. Add a hardcoded username and password to /login in express. If the user sends with the correct login, they should now have access to /protected. Also, make sure that users who are not logged in do not have access to /protected
2. Store the passwords in a database instead of in the code.
3. Add the ability for a user to register on the page.
4. Prevent users from registering with a busy username. Feel free to check it out ( <https://redis.io/commands/exists/>).
5. Instead of storing the passwords in clear text, store the passwords hashed and salted with bcrypt.
6. Require the user to have at least 8 characters in the password.
7. Set additional requirements that the password must contain at least one special character and one number. *Tip: write your own functions that determine this*
8. Prevent the form from redirecting to /login but only sending a reqeust to /login
9. Add the welcome text instead to “Welcome {username}!”
10. *Difficult: Add an extra field to the database that determines if a user is admin or not (* [*https://redis.io/docs/data-types/hashes/*](https://redis.io/docs/data-types/hashes/) *). An admin sees the text welcome admin!*

# JWT

**5.1** Make a JWT with header { "alg": "HS256", "type": "JWT"}, payload {"school": "jensen", "course": "web security"} and key: "secret" on <https://jwt.io/>

**5.2** Now do the same JWT without JWT.io and only with <https://www.devglan.com/online-tools/hmac-sha256-online>and <https://www.base64decode.org/>

**5.3** Implement a token-based login system. There should be a login page, a page available only to regular logged in users and one for admins.

# Environment variables and RBAC

**6.1** Replace your hardcoded passwords into environment variables in a previous project.

**6.2** Create a set of optional endpoints. You can use a previous project if you want. Ensure that some of these endpoints are only accessible to users with “superuser” privileges. Remember to link the authorization to permissions and not roles.

**6.3** Limit your endpoints further so that they can only be accessed between certain times.

**6.4** Shut down IP addresses that have made more than five incorrect requests from all endpoints. If you want to make it more advanced, you can change the limit to a maximum of five requests in a 30-second interval. Please check out <https://www.abstractapi.com/guides/node-js-get-ip-address>

# TLS

**7.1** In the following exercise, we will take a website live with HTTPS. The exercise will also require you to pay for a domain and a server. For example, a domain can be purchased via: [www.loopia.se](http://www.loopia.se) and a server via [www.digitalocean.com](http://www.digitalocean.com) .

1. Select an application that you want to take to production.
2. Create a linux server. I was running on Ubuntu 22.04. <https://docs.digitalocean.com/products/droplets/how-to/create/>Also select password as authentication method.
3. Make sure your domain points to your new server's IP. For example: <https://support.loopia.se/wiki/dnseditorn-a-och-cname/>. This often takes a few hours to work through. All the world's DNS servers must now update themselves for your domain to point to your server's IP.
4. Connect to your server via the command: ssh root@<your domain>.
5. You are now on your server. We first need to set up a production server. Express devserver is therefore not enough. <https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-22-04>
6. We will now fix a TLS certificate. <https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-22-04>
7. Install Node. <https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-ubuntu-22-04>
8. Tell Nginx to redirect requests to Express. You do not need to do the first step as you already have an application. <https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-22-04>
9. Success!

# OAuth 2.0

**8.1** Test a few different flows at: <https://www.oauth.com/playground/>

**8.2** Implement an OAuth flow using GitHub's OAuth app. You can use a previous application.

**8.3** Do the exercises here: <https://oauth.school/>

**8.4** Set up a free application at <https://auth0.com/>.

# WebSockets

**9.1** Do a code-along with https://www.youtube.com/watch?v=rxzOqP9YwmM&ab\_channel=WebDevSimplified

**9.2** The task is to solve a previous assignment for this course. I have cleaned out some requirements to reduce the scope.

Task

To demonstrate that you meet the course objectives, you must build a secure full stack

web app, using the techniques we have covered in the course. There should be an app for

secure chat.

Requirements / features

1. [G] Backend must have a database that saves users and messages.

3. [G] Login with username and password, and register new user.

5. [G] After logging in, the app should remember the user using JWT.

This means that if you reload the page, the app should start in logged-in mode.

6. [G] App should contain "public" and "private" channels (check out: <https://socket.io/docs/v4/tutorial/api-overview#rooms>)

9. [G] When the user has selected a channel, messages can be sent in it.

10. [G] Each message must state which user sent it, as well as

at what time. (days, hour, minute, second)

# XSS and CSRF

<https://portswigger.net/web-security/all-labs#cross-site-request-forgery-csrf>

https://portswigger.net/web-security/all-labs#cross-site-scripting