COMP 3920 Database Systems 1

Assignment 1 - SQL Injection Website

Introduction:

For this assignment you will be creating a sign up/sign in website using Node.js and MySQL for a user database and MongoDB for sessions. This assignment will review Node.js fundamentals and explore SQL, NoSQL and HTML injection attacks and how to prevent them.

SQL Injection, NoSQL Injection and HTML Injection attacks are a big problems and need to be addressed in our Node.js code. If left unchecked, they can lead to attackers destroying our database, revealing private and/or sensitive information, logging in as an admin without the proper credentials, and many more malicious attacks! As programmers, we need to be aware of how these attacks are performed, what code allows for them to happen (bad code) and what code is needed to prevent them (good code).

You will create 2 versions of your code.
Version 1 will purposefully be written poorly to **allow** for SQL and HTML injection attacks (unsafe version).

Version 2 will be written properly to **prevent** SQL and HTML Injection attacks (safe version).

For all queries to the database in Version 2 ensure you are properly escaping your inputs to prevent SQL Injection Attacks. For examples in Node.js: https://planetscale.com/blog/how-to-prevent-sql-injection-attacks-in-node-js

For Version 1, create a folder called: `V1_Unsafe\` for your Node.js program: `index.js`.
For Version 2, create a folder called: `V2_Safe\` for your Node.js program: `index.js`.

In a video recording demonstrate **both versions**. Run Version 1 and show that you can attack the database by injecting some SQL as part of your input. I suggest trying to run a command (or partial command) that will be non-destructive. If you inject a "DROP DATABASE;" or a "DELETE" command, you may end up losing data! Your video **must show that MySQL ran your command** (or partial command). Show, in your video, a console output or the database structure/data before and after to prove your command (or partial command) worked.
Run Version 2 and attempt the same type of attack as shown in Version 1. Show that your code in Version 2 prevents SQL Injection attacks.

Note: The advent of new AI technologies such as ChatGPT and GitHub Copilot has given programmers the ability to write some simple programs faster than ever before possible. However, as developers we need to be sure, that if/when we use these tools, that **all the code** we use in our programs is well tested, fully understood, and does what it was intended to do.
The use of ChatGPT and/or GitHub Copilot is NOT being restricted for this assignment, though, I caution you not to simply take the code any tool provides you without investigating if it is correct for *your* circumstances. Ensure that you know what your code is doing and why! Ensure you could and would write the same code if you didn't have access to these tools!

The basic elements of your site will include:

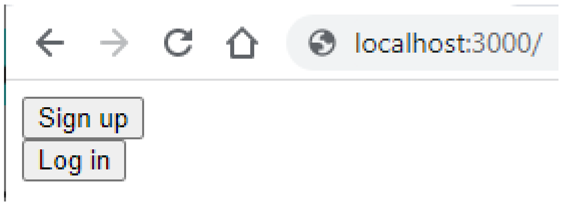
1. Login information is stored in a session.
    a. Store session in an encrypted MongoDB database (hosted on MongoDB Atlas).
    b. Encrypt session on the client side with a session key.
    c. Session lasts 1 hour.
    d. Ensure all member pages require a valid session (redirect back to the home page if no session or an invalid session is found).
2. A `.env` file for database passwords, encryption keys and other secrets
    a. Your `.env` is not included as part of the git repo (add it to the `.gitignore`)
    b. Add all environment variables to Qoddi, Render or other hosting service
3. Passwords are BCrypted before storing (passwords are never stored in plaintext).
4. A 404 status code is generated and a 404 page is displayed if a user attempts to go to a missing page.
5. A public folder for CSS, images, JS and other files is made available.
6. Your member database is a MySQL Database on FreeDB.net.
7. Your site is hosted on a service like Qoddi, Render, Cyclic.sh or other hosting service.

Objectives:

Your Node.js website have the following pages and functionality:

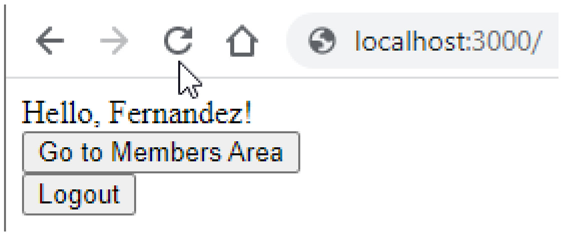
1. Home page – site: `/` method: `GET`

   If the user is **not logged in**, this page will have links to:

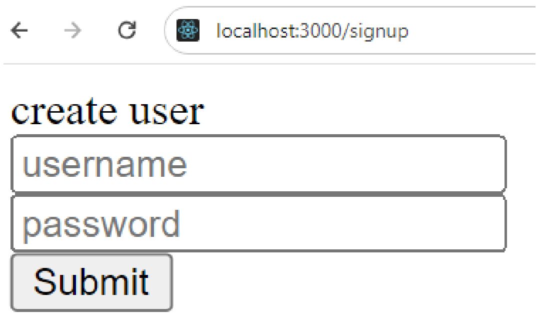
   a. Sign up (`/signup`)
   b. Log in (`/login`)

   Else, if the user is **logged in**, this page will:

   a. Say Hello and the name of the user – ex: Hello, Fernadez.
   b. Provide a link to the members area (`/members`).
   c. Have a link that will log the user out (ends the session and redirects back to `/`).

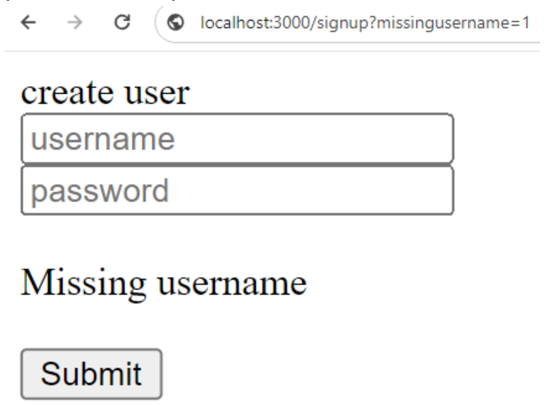
2. Sign up page – site: `/signup` method: `GET`

   A form will gather the following information for sign up:

   a. Username
   b. Password

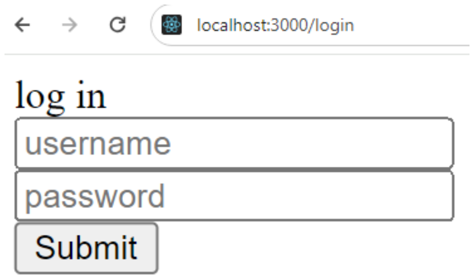
The signup form will `POST` the form fields.
You will validate all inputs to make sure they aren't empty. If the user has forgotten to enter one or more of the inputs, you will redirect back to `/signup` and display an appropriate message saying which field was missing. Ex: *Please provide a username.* HINT: you can use a URL query parameter to pass back to indicate which error message should be displayed.



Missing username

If the fields are non-empty, add the user to a user table in a MySQL database. Add the **username** and a **BCrypted hashed password**.
Then create a session and redirect the user to the `/members` page.

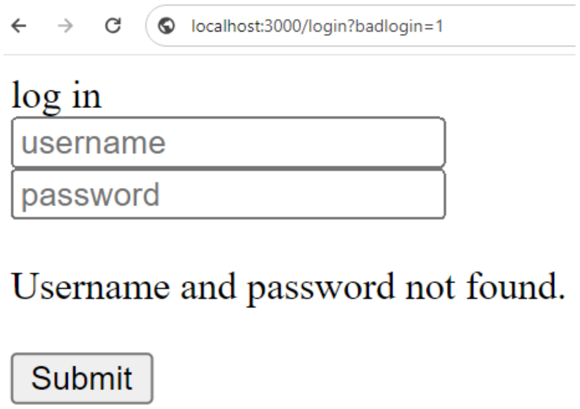
3. Log in page – site: `/login` method: `GET`



A form will gather the following information to log in:

c. Username
d. Password

The login form will `POST` the form fields. Check the user against the MySQL database. If the email matches a user in the user table, check to see if the password matches the BCrypted password. If the email and passwords match store the user's name in a session and log the user in (redirect to `/members`). If the log in fails, redirect to `/login` and display an appropriate message. Ex: *User and password not found.* HINT: you can use a URL query parameter to pass back to indicate which error message should be displayed.


localhost:3000/login?badlogin=1

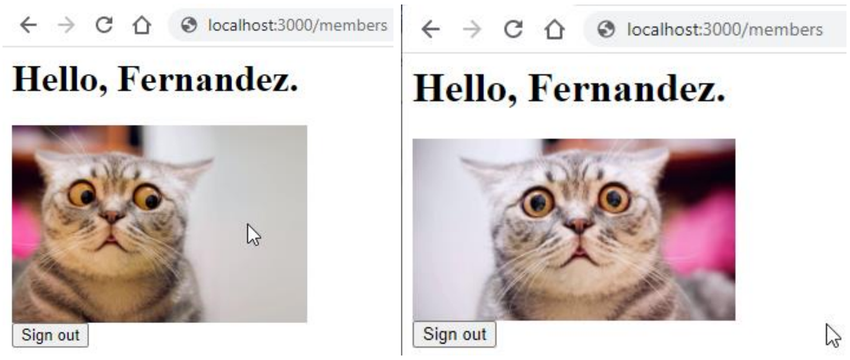
log in

Username and password not found.

4. Members page – site: `/members` method: `GET`

If the user has a **valid session** (is logged in):

a. Say Hello and the name of the user – ex: Hello, Fernandez.
b. Have a link that will log the user out (ends the session and redirects back to `/`).
c. Display a random image from a selection of 3 images.
   The images should be stored in a `/public` folder on the server.


localhost:3000/members

Hello, Fernandez.

localhost:3000/members

Hello, Fernandez.

If the user has **no session** (or an invalid one – i.e. not logged in):

a. Redirect to the home page (`/`).

Marking Guide:

Criteria	**Marks**
A home page links to signup and login, if not logged in; and links to members and sign out, if logged in.	5 marks
A members page that displays 1 of 3 random images stored on the server.	5 marks
The members page will redirect to the home page if no valid session is found.	5 marks
The sign out buttons end the session.	5 marks
All secrets, encryption keys, database passwords are stored in a `.env` file.	10 marks
A 404 page that "catches" all invalid page hits and that sets the status code to **`404`**.	5 marks
Session information is stored in an encrypted MongoDB session database. Sessions expire after 1 hour.	10 marks
Password is BCrypted in the MySQL database.	5 marks
Note: You will receive a **mark of 0** if your code doesn't run (i.e. too many errors to run properly, cannot connect to the database, etc.)! I will be running the code on your Hosted app, please make sure it is available until after you receive your marks!	
Subtotal:	**50 marks**
Video demo	
Shows Version 1 (unsafe version): Demonstrate an SQL Injection attack on the database where the user input causes a command (or partial command) to run. Show the effect(s) of the command (or partial command) run.	**50 marks**
Shows Version 2 (safe version): Demonstrate that the same type of attack, shown on Version 1, does not work.	**50 marks**
Total:	**150 marks**

Submission Requirements:

Submission:	**File name:**
A txt file containing the Hosted app for **Version 1 Unsafe** URL. ex: `http://xrb198fpq1a.us11.qoddiapp.com/`	Hosted_app_v1_unsafe_URL.txt
A txt file containing the Hosted app for **Version 2 Safe** URL. ex: `http://qa3p739n1rh.us12.qoddiapp.com/`	Hosted_app_v2_safe_URL.txt
A zip file containing **all the source code** required to run your Node.js applications (both the safe an unsafe versions). Do **NOT** include the `/node_modules/` or the `/.git/` folder	webite_site_code.zip
A txt file containing a Youtube video link to your video demo. ex: `https://youtu.be/dQw4w9WgXcQ`	Video_demo_URL.txt

Video Demo Requirements:

Your video needs to include:	**Example:**
Your name	Patrick Guichon
Which assignment you are demoing	Assignment 1

Notes about your video:
If I don't see all the functionality and features of your programs and attacks demonstrated in the video, you will lose marks for not having completing this part of the assignment.
You do NOT need to show your face while recording the video, however, you should be narrating the video. Tell us what you are doing, as you are demoing your programs and attacks.
Make sure we can hear your voice clearly, and at an adequate volume.
Max video length: 8m:00s. Your video must **NOT be longer than 8 minutes**.

Demo the following functionality in this order:
1. Show that your MySQL database has a user table, but that it is empty.
2. Show that your MongoDB collection for sessions is empty.
3. With **Version 1** of your Node.js code, add a user using the sign up page. Confirm that the user has been created in the MySQL database and that the password is encrypted. Confirm that the MongoDB database contains an encrypted session and the id matches the cookie in your browser.
4. With **Version 1**, click on Sign out. Show the session has been deleted from MongoDB
5. With **Version 1**, delete your cookie and attempt to go directly to `/members` without a valid session.
6. With **Version 1**, from the login page enter a valid email and password to show you are properly logged in.
7. With **Version 1**, go to a page that doesn't exist (for example `/doesnotexist`) to show the 404 page.
8. With **Version 1**, perform an SQL Injection attack on either the sign up or sign in pages. Show the effect(s) of the execution of the injected SQL command or partial command.
9. With **Version 1**, perform an HTML Injection attack on the sign up page by creating a user that contains some HTML (or Javascript). Log in as that user and show that you page runs the HTML (or Javascript).

10. With **Version 2** of your code, attempt the same type of SQL Injection attack as before in Version 1. Show that the attack is prevented in **Version 2** (the SQL command or partial did not execute).
11. With **Version 2**, log in as the user created from Version 1 that contains HTML (or Javascript). Show that the HTML tags are properly encoded and displayed, instead of being treated as code (tags).
12. With **Version 2**, show the code for your site in your editor: - Show that you are using variables from your `.env` file. - Show that your sessions are set to expire after 1 hour. - Show that you are parameterizing user inputs using `namedPlaceholders` in your MySQL queries. - Show your code for the 404 catch all route.
13. With **Version 2**, show that your `.env` is added to your `.gitignore` and your `.env` is **NOT** in your git repo.