CS 7062 Report

Contents

[CS7062 Project Analysis 2](#_Toc133432875)

[Entity Relationship Diagram of Database 6](#_Toc133432876)

[URI of Video 7](#_Toc133432877)

[High Level Description of Website 8](#_Toc133432878)

[Instructions for running website. 9](#_Toc133432879)

[References 10](#_Toc133432880)

# CS7062 Project Analysis

Development Approach

In terms of approach to my project, it was originally haphazard. I designed pages for the various features of the site, to be further expanded upon later. For example, I made a login, home and search page. I also found a CSS style, PureCSS, and decided to use it alongside my own CSS to further develop the HTML pages I was developing, because I liked its buttons and scrollable navbar. I created a very basic template for a web page and used that as the basis of all the pages in my site. As Node was central to what I was doing, the standard template was an EJS file. I knew CSS was a particular weakness of mine and so creating a particularly beautiful site was never my focus, instead I tried to make it quite functional to compensate for my struggles with responsiveness and modern design

The first step to my website was creating a database, as most of the website’s function was reliant on a DB, which I implemented through XAAMP and MySQL. I was aware that the DB was likely going to rely on some degree of normalisation, vital to avoid redundancy.[[1]](#footnote-1) I tried to adhere to the Normal Forms as defined by Edgar Codd, as such each row of my database must be uniquely identified, as according to the 1NF, according to the 2NF every non-key value in the table must be related to the name of the table so for example the user table had no business including information about the albums they reviewed. Instead that would be kept in its own separate table. The 3NF is the hardest to stick to and asserts that all fields of the table must be determined from the primary key. Many to many relationships were also vital for allowing certain functionalities of the site, for example if I wanted to create track lists for my albums it would be impossible to properly search through them using a standard song list field in my album table. However, through foreign key relations It was possible to create a song table and a table that linked to both it and the album table allowing SQL commands to ask for the info held in both tables inner joining based on the middle table. SQL however was a big part of my issue approaching this project, as it can be difficult to figure out exactly the best query to use. Thankfully with the use of axios and a REST API once a query was found to be useful it was quick and powerful it its application.

Security was the first major avenue I tackled, to create some protection for my website. I was aware that the security of the website was important, after all some “56% of all internet security” is from bad actors.[[2]](#footnote-2) With reference to a lab in CS7062 I knew I wanted the DB to check both the username and password before they could enter and create a session for them. By using SESSION authentication, and calling out to an SQL database for credentials, it rendered the system reasonably secure. And only allowed unregistered users to wander wherever they were allowed to be. I realised later that there was still a very large problem, that being that the passwords were stored in plain text. This rendered users very vulnerable to any attack that could get access to the DB and see them, so I used ‘bcrypt’ to hash the passwords when they were entered in the register and stores those hashes in the DB. Whilst not immune to any bad actors, I felt this made my website reasonably secure.

Originally, I had debated between using APIs or not in my website, eventually I did use them although not for everything. I used a simple REST API, whilst mostly used as a medium to complete DB queries a REST API also has use in terms of its scalability, and flexibility.[[3]](#footnote-3) The speed of their queries is also useful. Combined with the EJS notation I would be using in my webpages, and it became very clear just how useful they were for keeping my queries. I did have some problems however with fully understanding how they worked, and quite often I felt my own struggle with SQL neutered the power of them. However, they did end up proving useful in the construction of the website. I checked their usage through postman, and eventually did most of my tests through axios itself to check that my gets and posts were going through. My inconsistent use of parameterisation of the terms fed from the API to the DB, as it was more to fix issues with certain queries not working properly meant that whilst mostly protected against SQL injection attacks the DB was not completely so.

When it came to approaching the webpage EJS was another necessity, as it effectively allowed me to create HTML pages but with all the functionality, I could need for the website I was making. EJS itself has many advantages, such as its scalability.[[4]](#footnote-4) This is important, as although my website is very simple in the future websites, I work on may not be. I also had somewhat considered using React however I realised this would be a poor choice due to my unfamiliarity with that stack. I also felt that EJS was more useful for a website that did not overall have too many things to render. The for each statements of Node, and EJS’ ability to render them live on a webpage were a necessity to output the objects returned by the REST API routes I was using, as they are not indexed like an array.

In addition, Visual Studio code and the various Node dependencies were also vital to the development. I needed the ability to interact with the DB and the extensions Node.js has were vital. Having a user-friendly interface to work with these dependencies was therefore quite useful.

In conclusion, the website was quite difficult for myself to make which I would chalk up to my struggle and unfamiliarity with DB queries and getting them to perfectly mesh with what I wanted to render in my JS. In addition, I struggled tremendously with making my website pretty and responsive. As it stands however the website is reasonably functional and fulfils the requirements such as allowing anyone to search for albums or member collections, filtering based on genre, rating and artist, you can register as a member and members can create and manage their own collections as well as leave reviews on other member collections or regular collections. I did however make some compromises, as I allowed users to insert some demographics and additional user info, like age and a paragraph that describes them, but I didn’t allow them to insert their gender as I was not sure how to best approach categorisation of gender and instead accepted it as a limitation and left it. In addition, I also did not give the ability for users to delete from their collections, which is something that could do to be improved on in a future iteration of the website.

Bibliography

Microsoft Learn, "Description of the Database Normalization Basics." microsoft.com. https://learn.microsoft.com/en-GB/office/troubleshoot/access/database-normalization-description (accessed Apr. 21, 2023).

Agnes Talalaev, "5 Reasons Why Website Security is Important." patchstack.com. https://patchstack.com/articles/reasons-why-website-security-important/ (accessed Apr. 21, 2023).

Chakray, " What are the advantages of a REST API?"" chakray.com. https://www.chakray.com/advantages-of-rest-api/ (accessed Apr. 22, 2023).

Roaring Studios, " Why Node js | Advantages of EJS |?." roaringstudios.com. https://roaringstudios.com/blog/advantages-of-ejs (accessed Apr. 22, 2023).

# URI of Video

https://youtu.be/\_RDm1YodYbs

# Graphical user interface, application Description automatically generatedEntity Relationship Diagram of Database

# High Level Description of Website

In terms of code the Website functions off the use of two ports 3000, and 4000, with the use of a REST API to serve the various requests made by the website to the database.

The login side of the website does not use an API, instead directly calling DB query to render the login and post the data it receives to the DB. Through use of bcrypt.js the password they enter is compared with the stored hash and validated. Once validated, the DB finds a row with their info, a session is created with the authen being their id which will be used in other queries.

The register page allows the user to insert their information into the DB, effectively creating an account with the website. The password they choose is hashed using bcrypt, and that hash is then stored in the database, whilst bcrypt keeps a hold of what the original password was allowing it to be compared to in the login.

The home page is served using app.get and relies on a DB query to serve its info to the user, if their session is not authenticated, they are redirected to the login screen. It also prints a greeting, which’s space is use later for other requests.

Adding to the user collections, as well as the regular albums is also reasonably simple use of axios and API. The forms are posted from the webpage using axios, and then the API receives and inserts the values into the database’s user\_album table. The page is then rendered again with a message informing the user that their album’s name was added to the database.

Creating a user collection relies on using axios to get the page that allows you to fill out your review, and then post it to the DB with use of the API. The information taken first inserts into the DB the review, before then making a query to update the upvote count of the collection before finally making inserts into the useralbum\_review table which links all reviews to the user collections. The same process is employed for reviewing a regular album, with some changes in SQL queries and the table used.

The search function is split in two, one for user collections and the other for official albums. It relies on several API calls being made, namely a posting of information that is optional and dynamically expands to include the three search queries used. It then renders a page where all the results are displayed. Each has an inspect button, which allows the user to go to another page which outputs the information of the album that has that id, with the inspect button holding the ids of all the albums outputted using the row id as the query. A very similar technique is used in managing of the user album, where they are allowed to see all reviews of collections they made.

The process of adding reviews is also superficially rather similar to adding an album, with separate processes to add reviews to user collections or official albums. With the info posted through axios, the API inserts the review into the DB and chooses which table to associate the review with be it user collection or official album.

# Instructions for running website.

* XAAMP for use of Apache and MySQL, other options include MAMP, the mac equivalent.
* My DB must be imported into an SQL database, and its name must be “cs7062project”, this can of course be changed if you are comfortable editing the code of the app.js and changing the path of the server you are on
* The DB you use must be on port of 3306, as is typical for a XAAMP’s DBs. The host is set as “localhost”, the user as “root”, the password is blank, and the database is set as the name above. If the DB you are using is on a different port with different values you must edit both the .env file and mysql.createConnection variable to reference your DB otherwise it will not work.
* NPX `nodemon` is to be used to start the server, and should be launched from the terminal, it is also recommended you use some form of editor or gui to display the code and so you can see the console logs, if they appear, as such I recommended using Visual Studio Code. Due to Visual Studio Code being able to work with so many various languages it also has utility for seeing the various dependencies in action.
* The various dependencies will be included with the file, so there is no need to install them. The dependencies used will be listed later in this report.
* The website itself is hosted on http://localhost:3000/, and the API on http://localhost:4000/, if you are curious about any API routes visit 4000 otherwise stick to 3000. The default route of 3000 is the index page which provides a sort of heart of the system, the rest of the website is self-evident from this page.
* Only certain functionality of the website is allowed to a user who has not registered. You may search and read the information of the albums the search returns, but other than that you can do very little. To access the home pages and the additional functionality of the website you must log-in.
* A user created for you will not exist, so you will head to the register page and fill out all its fields, making use of the select options as well to select your user type. Once this is done it will redirect you to the login, where you will be able to login with your new username and password.
* Failed logins will redirect you back to the page, so if you do not get in it is a sign you are either not registers or have misspelled your details.
* Registered users get access to the ‘Home’ page, which contains the core functionality of the website provided for them. Here you can choose options like making a personal album, managing your albums, leaving reviews on official albums or community collections. All these pages are protected by the authen of your session, so if you are not logged in you will be redirected from these pages back to the login page.
* There is an additional button called home on the navbar on certain pages allowing quick and speedy return to the home when you are done with the page you are on.
* This concludes the instructions of how to run the website.

# References

Dependencies:

* Axios 1.3.5
* bcrypt 5.1.0
* bcrypt.js 2.4.3
* cookie-parser 1.4.6
* dotenv 16.0.3
* ejs 3.1.9
* express 4.18.2
* express-session 1.17.3
* mysql 2.18.1
* nodemon 2.0.22
* <https://purecss.io/> (PureCSS)

External code snippet sources / inspirations for code/markup

<https://csc7062.gitbook.io/lab-11/> (Rest API and dependencies needed)

<https://csc7062.gitbook.io/lab-10/> (API and Node and dependencies needed)

<https://csc7062.gitbook.io/lab-09/intro-to-sessions/login-set-up> (Sessions and dependencies needed)

<https://csc7062.gitbook.io/lab-08/mysql-gets/node.js-+-express-+-mysql> (Working with DB from node.js and dependencies needed)

<https://web.archive.org/web/20131230143316/http://blogs.msdn.com/b/sqlphp/archive/2008/09/30/how-and-why-to-use-parameterized-queries.aspx(Idea> (idea of parameterisation and arrays to use them)

<https://csc7062.gitbook.io/lab-07/start/more-templating> (node.js functions and working with express and dependencies needed)

<https://csc7062.gitbook.io/lab-03/> (CSS ideas for basic responsiveness and modern design)

<https://www.npmjs.com/package/bcryptjs> (installation and usage of bcrypt)

<https://github.com/Vuka951/tutorial-code/tree/master/express-bcrypt> (tutorial of using this in a DB.)

<https://www.w3schools.com/js/js_arrays.asp> (assigning object to an array)

<https://www.w3schools.com/js/js_array_const.asp> (assigning values to array)

<https://purecss.io/> (How to use some of PureCSS’ features.)

<https://purecss.io/menus/> (Use of scrollable navbar for some responsiveness at lower sizes

<https://www.w3schools.com/cssref/css3_pr_justify-content.php> (For my attempts at justifying the CSS)

<https://www.includehelp.com/node-js/ejs-if-else-statement-ejs-conditions.aspx> (using EJS if else statements.)

1. Microsoft Learn, "Description of the Database Normalization Basics." microsoft.com. https://learn.microsoft.com/en-GB/office/troubleshoot/access/database-normalization-description (accessed Apr. 21, 2023). [↑](#footnote-ref-1)
2. Agnes Talalaev, "5 Reasons Why Website Security Is Important." patchstack.com. https://patchstack.com/articles/reasons-why-website-security-important/ (accessed Apr. 22, 2023). [↑](#footnote-ref-2)
3. Chakray, "What are the advantages of a REST API?" chakray.com. https://www.chakray.com/advantages-of-rest-api/

   (accessed Apr. 22, 2023). [↑](#footnote-ref-3)
4. Roaring Studios, " Why Node js | Advantages of EJS |?" roaringstudios.com. https://roaringstudios.com/blog/advantages-of-ejs (accessed Apr. 22, 2023). [↑](#footnote-ref-4)