Web platform development 2

Coursework 2

“I declare that all work submitted for this coursework is the work of Ammar, Lewis, Anthony and Ryan alone unless stated otherwise.”

Group:

[LMCKAY202@caledonian.ac.uk](mailto:LMCKAY202@caledonian.ac.uk) S1622906

[AMAHMU201@caledonian.ac.uk](mailto:AMAHMU201@caledonian.ac.uk) S1714878

[ADOYLE205@caledonian.ac.uk](mailto:ADOYLE205@caledonian.ac.uk) S1427055

[rhughe205@caledonian.ac.uk](mailto:rhughe205@caledonian.ac.uk) S1918370

Link Design

The URL’s shown below couldn't be implemented, however the designs of what they were supposed to look like are shown. For the implemented site we used Localhost:3000/,

Localhost:3000/login, Localhost:3000/register and Localhost:3000/guestbook, the group discussed the designs below for what would have been a part of the project.

The URL designs that the application would have responded to is :

<http://localhost:3000/login/195#2332> - The response that is given for the page is it takes the user to the login page, where they are presented with an input field to insert their username and password. If the user gives an unidentified username and password then they won't be able to proceed to the coursework page. If they do have one then they can proceed.

The path for the url is /login as it is easy and human readable for the user, the query is shown with the fragment as that as it was neater to display in the url window.

<http://localhost:3000/register/212#342324>- The response that is given for this is the user is taken to the register page in which they are tasked to make a username and password, the password is set to be a minimum of 8 characters to reduce potential duplicate passwords and security breaches that the site may have. If the user does not make a password that is 8 characters then the user cannot register an account and is unable to proceed to the login page that will allow them to access the guestbook page. The path will be /register as it seems like the logical name for the path as the login page has the path /login. The query string and fragment are also detailed as well, but it is set like that to hide the functionality and to make sure no private details are shown in the url, this is a common example of bad practice.If private details are shown in the URL.

<http://localhost:3000/guestbook/232#234234> - The response that is given is that the user can see what other people have placed; this includes the title of the coursework project, the module for which is to be submitted. A breakdown of milestones for each coursework project, an intended due date and the actual completion date. When the user is logged in the coursework projects can be added, removed, modified and it shows a list of incomplete courseworks and their milestones. The path is named /guestbook as it was the suitable name for it, the query string and fragments are there the pull request number and the fragment being the scrolling position.

The web page starts on the default page where the user can choose between logging in and registering, when registering is picked and details are done they are sent to the login page, this is then done and transferred to the guestbook page where the users can use the functionalities the page offers.

After the domain the path can be controlled by the buttons on the page until the user closes the web page manually.

When the app is displayed in heroku the domain and protocol are suited to what heroku generates making the url more neater and makes it look less like an ip address. This is done when the application is run in heroku.

Persistence

The website requires a database to store users’ details for login and then the information for the users coursework. We have chosen to use a nedb database in our application. This type of database can be both persistent and used as in-memory storage. The login and register side is persistent in session the database is populated by a method in the model “init()” temporarily every time the application is run. The code in the users.js contains the lines to activate the database for this section telling the application that it is a nedb database “const Datastore = require("nedb");” . Data is sent to the database via the post function in the controller when the user inserts registration data “controller.post("/register", function(request, response) ”. This method also checks the database for the username entered to check the user doesn’t exist already. This data will be stored to the database and will allow the user to login. Since our database is only persistent in session if the server was to be stopped and started back up the users details would be gone from the database, the database only keeps the new data stored while it is running. This way, for short term it works well for something small like this project, but is not future proof at all and would need to be changed. It is a feature we understood and wanted to put it but couldn’t due to time constraints. To have this application to run in real world persistence would be a key element that needed implementing to make it more usable.

With regards to the database, the entries within the database are unable to persist currently between sessions, though this is something we are aware of and would have liked to have fixed should more time have been available to us to work on it. To make the database persist would have required a database file with which to write the logs from the database to the file. This is done by specifying a path to the database file within the program. Currently not having a persisting database made testing easier for us, by reducing resources used and time to boot the website but meant long term usage was no longer able to be done.

Test Reports

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test I.D** | **Action** | **Expected outcome** | **Status** | **Data** |
| #001 | Localhost:3000 | Landing page loads | As expected |  |
| #002 | Login button clicked | User taken to Localhost:3000/login | As expected |  |
| #003 | User enters invalid username OR password | Return to login user doesn’t exist | As expected |  |
| #004 | Username already exists | Return to login | As expected |  |
| #005 | User enters valid username and password | Access granted to user and they are taken to  Localhost:3000/courseworks | As expected |  |
| #006 | User clicks register button | Taken to  Localhost:3000/register | As expected |  |
| #007 | User enters a username that already exists and password 8 or more characters long | Returned to login | As expected |  |
| #008 | User enters valid username and password under 8 characters | Message displayed to user “password is too short” | As expected |  |
| #009 | User enters a username that already exists and password under 8 characters | 2 messages displayed “username already exists” and “password too short” | As expected |  |
| #010 | User enters a valid username and password 8 or more characters long | User is taken to login page and users details are added to database | As expected |  |
| #011 | Localhost:3000/courseworks | User is taken to a listing of all of the courseworks | As expected |  |
| #012 | Click add courseworks button | Taken to page to add coursework | As expected |  |
| #013 | Submit courseworks details | Add courseworks to database and courseworks displayed | As expected |  |
| #014 | Click edit coursework | Taken to page to edit coursework | As expected |  |
| #015 | Edit module and both dates, then click submit | Details changed on coursework listing page | As expected |  |
| #016 | Click delete coursework | Coursework details are removed | As expected |  |

Summary test results

The tests that were done on the site worked as expected, The site itself took numerous attempts for the tests to be completed and done, but as shown in these results they worked as expected. Out of all the tests done (16 out of 16 tests) all of their functionality works as expected from the test case that were completed. These were tested on the login, register and coursework pages as well as the landing page to test out the functionality that the pages offer.

Application Security

An application's security is one of the most important features, for a software to have good security it must be capable of protecting the site data including the internals of the site as well as protecting any of the users data. One of the most important pieces of data that is stored are users passwords. If hackers are able to gain access to the database that stores all of the user details, or are able to intercept this information it would be detrimental to the user’s account, not only on the software you are developing but on other sites where they might have used the same registration data. Allowing hackers to gain access to these databases which should be secure would make user’s lose trust in you and would stop using your application.

One way to protect user’s passwords is by using Bcryptjs. Bcryptjs can be installed as a node app using the “npm install bcryptjs” command, it allows for passwords to be hashed adding an extra layer of security onto the system. This is done through a process called salted hashing, which is the generation of completely random bytes and by combining it with the user's passwords, it creates a unique hash according to what the user has their passwords set to. A major benefit to using this type of hashing is it will prevent rainbow table attacks which is when hashed passwords are reversed using common hashing algorithms, however this becomes impossible once the passwords have been salted. The process behind this works as follows: First the users password is hashed and stored into the database, upon a login attempt the hash of their password is checked with the one stored within the database, if it matches then the user will be able to login successfully, if it doesn’t match it will send them back to the login screen with an invalid credentials message. Bcryptjs gives us complete control on the values of the salt allowing us to set a value to the number assigned. The higher the number of values assigned, the more secure the data will be as it will be harder for a hacker to process all the different combinations, with a higher number of values the time it will take to process the login attempt will increase because of all the values that have to be processed, this means it’s important to balance the time it takes to process and the number of values assigned, as to protect the users and allow for a good user experience.

There are two important parts of application security that should be present within any application, authentication and authorization. When considering application security the group discussed the usage of Passport for authenticating logins, however it was not implemented fully as the objectives of the project changed during the change in circumstances, which resulted in our project no longer being externally hosted. However, the application developed does have a register and login feature which allows the user to register for an account and login with the registered details, upon any login attempt the details will be crossed check with the details stored in the database, if the application finds those details and they match, then the user will be able to access the application, in the case those details cant be returned the user will be returned to the login page and asked to login again. This is how the concept of authorization was considered within the project, by ensuring only registered users can access the application.

As for what security could be improved within our application, ideally a form of logging could be implemented in order to track who has accessed data. This is essential incase of a security breach within the system as it can log who has access to which data and who doesn’t. Ideally, in order to ensure security our application would be tested using different methods such as static testing or dynamic testing, the group wanted the application to go through static testing using a tool called SonarQube which performs static code analysis and would help to bring up any security problems or if there is any part of the applications code that is improperly used and could lead to security vulnerabilities.

Overall, application security was an important consideration within the production of the application, as it plays a key role in the users experience and the main defense of the system. Within the scope of the project, it was not necessary to have as many security features as an application developed for a business would and due to the project being carried out on a shorter deadline the team had to decide which security features would be implemented and which one’s would be left out, this would be similar to how a real application would be developed and the same discussions about system vulnerabilities and the best way to protect user data would be carried out.

Changes to the system due to the Circumstances

When implementing the project, the change in circumstances which suddenly occurred resulted in changes to the initial design, this was because what was originally wanted in the designs became unrealistic to implement given the circumstances that the group was working under. In order to work with these circumstances our group communicated regularly on an application called discord, an instant messaging service, allowing us to track the progress of the project. Through discord we provided daily updates and uploaded any files that team members had to change and it was useful in keeping every team member up to date with any changes that were occuring.

Although we had a form of communication using Discord, the change in circumstances meant it was impossible to meet in person to work on the project. When everything for a project is based online especially with something that the group members were not as familiar with, it made things a lot more challenging, one particular challenge was when a problem occurred within the application as we had to resolve it through messaging each other using Discord and it meant a lot of time was spent troubleshooting issues. Another problem stemming from the change in circumstances was that it was no longer possible to have in person meetings, all meetings had to take place online and this resulted in an unclear schedule for all the work to be undertaken.

If there were to be any benefits from the change in circumstance, it would be that due to the university being canceled, it left the group members free to work on the project at any time they wanted, this does not mean however that there was an abundance of time for the group as many group members had obligations outside of university that resulted in lossed time for the project. Also, due to the group members distance from the university, travel time was essentially lossed time that could be used for working on the project, this was negated due to the change in circumstances giving a little extra time to work on the project.

As discussed within the Security section, there were several features that we had initially planned for and hope could be introduced into the project, which became impossible later on, as we had to be realistic with the time schedule we were working with and the change in circumstances that occured, thus several features had to be removed post-planning. Overall, the change in circumstances negatively affected the progression and end result of the project.

# **Equal Share of Workload Declaration**

We, the undersigned, hereby declare that we have shared the workload for coursework 2 of the module Web Platform Development 2 equally.

Team ID:

Team member name: Lewis mckay Signature: lewis mckay

Team member name: Anthony Doyle Signature:Anthony Doyle

Team member name: Ryan Hughes Signature: Ryan Hughes

Team member name: Ammar Mahmud Signature:Ammar Mahmud