**Web Platform Development 2 – Coursework Group Report**

**By Ellis Holt & Alastair MacGregor**

*I declare that all work submitted for this coursework is the work of Ellis Holt & Alastair MacGregor alone unless stated otherwise.*

**Introduction**

This is the group report for the Web Platform Development 2 module. This report will outline the group project in 4 main sections: the link design, the persistence mechanism, the test report, and an appraisal of the application security. A declaration of the work completed is on the title page of this document.

Each section will have a detailed breakdown explaining the thought process behind the actions taken as well as the result of the overall application. This project was undertaken by a team of two. Communication was maintained throughout the development of this project, even though team members were not able to meet up in person to discuss the project due to the coronavirus outbreak.

**1. Link Design**

The URL design of the application was completed with readability, maintainability, and scalability in mind. This means that the links can be easily read by the user and give the user an idea of what each page does, and enables them to easily keep track of what page they are on and the overall functionality of the application.

Each link is made to be as understandable as possible to the user with the overarching design being done in a way that leaves room to add further functionality as the application grows in size and complexity. Maintaining human readability is useful because as the website could become potentially more complex in layout, the readability element helps the user keep engaged if the website increases in scale. This could be particularly useful in future if the application expanded on features with more pages. An example of a further page could be a password recovery page, which would be an extension of the login process as it would most likely be executed from the login page.

This is delivered by keeping the links relatively short. By decreasing length, this decreases complexity as the user has less elements to read or decipher. Furthermore, each link contains a one or two word breakdown of the functionality of the page, which are relatively obvious in their purpose. E.g. auth/login could enable the user to relate to the words authorisation and login, which will help them understand that they are on the login page.

There is no search functionality in the application, therefore there are no search items that can be displayed in the session URL. The URL’s also show limited personal data, except for where it may be relevant. For example, when the user shares a project, the user’s username will be displayed, helping the reader to understand who’s project it belongs to if the link is shared.

|  |  |  |
| --- | --- | --- |
| **Path** | **Functionality** | **GET/POST** |
| / | Welcome page of the site with login/registration options | GET |
| /auth |  |  |
| /auth/login | Render login page | GET |
| /auth/login | Authenticate user by passing their username and password into Passport Strategy | POST |
| /auth/register | Render registration page | GET |
| /auth/register | Creates new user and stores them in the users database | POST |
| /auth/logout | Logs user out and redirects to application home page | GET |
| /users/:username | Renders users list of coursework’s based on their login information | GET |
| /users/add | Renders page that allows users to add new coursework | GET |
| /users/add | Stores new coursework in database | POST |
| /users/:username/:coursework\_title | Renders chosen coursework an individual page that can be shared to unauthenticated users | GET |
| /users/:username/edit/:coursework\_title | Renders ‘edit’ page with chosen coursework information rendered in form | GET |
| /users/edit | Stores coursework changes in database | POST |
| /users/delete/:username/:coursework\_title | Deletes chosen coursework from database | GET |

**2. Persistence**

The application allows users to register with a username and password, this information is then stored in the database for future logins. After creating an account, users are allowed to Add courseworks to their profile, these courseworks are then stored in the database. These courseworks can then be modified, shared and deleted.

The modification of coursework is done through retrieving the data of the selected coursework from the database and redirecting to the ‘edit’ page with the fields pre-populated with the relevant information, ready to be modified.

Sharing of the coursework required retrieval of the selected coursework information from the database, this was then displayed on a new individual coursework page that didn’t require authentication to access as required in the specification.

Deletion of coursework required finding the chosen coursework in the database and removing that object from the array of courseworks.

**3. Test Reports**

The application had lots of input fields for the users to input their data, therefore each field had to have appropriate input validation applied. Such as dates being in a DD/MM format. This allowed the system to be able to work more seamlessly with the data as it was aware of the format and characters to expect. The ‘rules’ applied to inputs are detailed below:

|  |  |
| --- | --- |
| Field | Format |
| Coursework |  |
| Coursework Title | * Length Range 1-20 |
| Module Name | * Length Range 1-10 |
| Due Date | * ‘DD/MM/YYYY’ Format * Fixed Length – 10 characters max * Allows only numbers and ‘/’ |
| Milestones |  |
| Title | * Length Range 1-20 * Can only contain numbers, spaces and letters |
| Description | * Length Range 5-150 * Can only contain numbers, spaces and letters |
| Due Date | * ‘DD/MM/YYY’ Format * Fixed Length * Allows only numbers and ‘/’ |

**Unit Tests**

Unit testing was achieved by checking that data was loaded onto the database. Initially this was done through using MongoDB Compass on a locally stored database. When each value was entered, it was checked that it was entered onto the database. This was then continued after the database was moved online.

**System Tests**

Compliance with requirements – after unit testing had taken place to test individual components of the application (all aspects of data storage and modification), testing began on weather or not the application was developed in line with the requirements. The requirements of the application are laid out in each page of the application:

**Test Scope:**

* Testing of login page (user account requirement)
* Testing of registration page (user account requirement – new user)
* Testing of coursework page (view coursework requirement)
* Testing of sharing page (share feature requirement)
* Testing of adding coursework page (adding coursework requirement + deleting coursework)
* Testing of modifying coursework page (modifying coursework requirement

**Input Tests:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Action** | **Expected** | **Actual** |
| Landing Page | N/A | Presentation |  |
| Input Login | N/A | Presentation |  |
| Login fail | Wrong details | Message |  |
| Login Attempt Wrong | Missing details | Message |  |
| Login Attempt Correct | Correct details | Taken to coursework page |  |
| Registration | Registration Button click | Presentation of page |  |
| Registration empty details |  |  |  |
| Registration success | Correct details | Login available: |  |
| Registration wrong | Bad password | Message |  |
| Modify Coursework | Allows user to change pre-existing coursework from coursework page |  |  |
| Add CW | Correct details | CW added |  |
| CW Modify Display | Correct CW Displayed | Display data coursework with editable fields |  |
| CW Share Display | User clicks on share button, gets taken to this page that can be shared | CW Example |  |
| CW list page | User is taken to this page after login. After Coursework is entered, they should see a list along with button options: | Shows CW list |  |
| Remove CW | User can click bin button to delete existing cw | CW removed from list |  |

**Summary Of Test Results:**

The test results show that the code fundamentally works. This enables the user to login to the application, add courseworks, remove courseworks, and edit courseworks. All of the designed functionality works as expected. Since an iterative approach was taken to develop this application, bugs and errors were continuously removed throughout. This meant that if features were not working, they were removed. The test scope was designed to ensure that functionality of the application works as expected.

Ultimately, the application works as a whole as designed and the original features were delivered upon as explained in the group presentation. The test functionality is better used in person. But the test table outlines each of the core functionalities of the developed application in the test scope, and each of those features have been delivered upon, delivering a functional application intended to be delivered from the initial group presentation.

**4. An Appraisal Of Application Security**

* **Account Creation**

For the purposes of this application, it was necessary to allow users to register in order for them to create their own coursework’s that they could keep track of. The registration process was done using a simple username and password. As with any sensitive user information, it must be kept secure.

This was done by encrypting the passwords via hashing. The built-in Node Crypto library was used to encrypt the entered password using a sha256 hashing function, following this the password was securely stored within the database. Doing this ensures that, even if the database is accessed by an unauthorised user, the account passwords are stored as their hashed version, therefore providing no value to the unauthorised user.

Improvements could be made to the security of the account creation part of the application. This would involve registration via email, allowing users to recover lost passwords, as the application in its current state lacks this functionality.

* Sessions:

The application makes use of sessions to enable the user to login and logout as well as see their data. This is necessary to protect data. For example, the data cannot be seen without logging in first, protecting data.

* User generated input:

The users generate input for the application. This contains details such as usernames, passwords, and other values for text fields. This also means that data can be exploited if it is inputted incorrectly. That means that data needs to be checked. To reduce the chance of SQL injection attacks, data can only be inputted the correct way. This means that all input fields are checked and limited and tested to make sure that database data cannot be manipulated to cause problems.

* Potential Future Features:

These are features which could be implemented to aid security but could not be developed in the original time frame envisage. Features that could help user security are features that are like: the implementation of a password and email validation system. This would mean that in order for the user to register they would need to connect their account to an email address to make sure that they are not a robot.

Another potential features is an account lock function. This is when the account locks itself down after the user has incorrectly entered their password for a certain amount of times in a given instance. This would prevent password bashing algorithms continuously guessing user passwords by brute force so that they can gain entry. This is an important feature as if it would not be implemented into the final application, the account security would be severely compromised. In theory anyone could gain access to this account through brute force attacks if there were no account lock feature. This would obviously require a password reset feature through email to recover their account.

Two Factor Authentication could also be used to enforce security further. The user could request that a code be sent to the user to ensure that the application has a secure login. For example, if the user logs in, a code could be sent to their phone to enable them to login. This would be good as only the owner of the phone would be able to login to the account.

A limited sharing option: whilst user data can be shared, to make it more secure, the user should be able to define who it is being shared to. This could prevent external operators gaining unwanted knowledge of student subjects and could therefore locate their classes using university timetables. This is a potential breach of privacy. Therefore the user should be able to, in future, define who they only want to share their data and projects with.

Another potential security feature that could be added is a timeout of sessions. This means that after a given amount of time of inactivity, the user is automatically logged out of their account. This means that the user won’t be able to leave their account forever logged in to a certain machine. This would enable passers by to gain unwanted access to an account of the user leaves their machine unprotected for a certain amount of time. A way that this could work would be after 10 minutes of inactivity, the application could automatically take the user to the login screen. However this must be done in a way that is not detrimental to the user. For example, if the user is entering data into the application, and they are logged out when they are away, the data that they are entering should be automatically saved to the database if they are logged out when they are away. That means that if they are logged out their progress will not be hindered.