Web Application Coursework 2

Statement of purpose:

The main purpose of the website that I have created is to allow users to be able to return and borrow books from a digital library. The project “Library Management System” is developed in flask, which mainly focuses on basic operations in a library like adding new books, and updating new information, searching books and members can return books. This project of “LIBRARY MANAGEMENT” of gives us the complete information about the library. We can enter the record of new books and retrieve the details of books available in the library. The website allows us to issue the books to users and keep track of when they were issued. The user can search for a specific book in a search bar, and the details of the book are displayed- like the title, genres and the authors. If logged in, the website allows the user to borrow a book from the list of available books. The system also has an admin panel, which allows a special user- the admin to edit the entries in the database- the admin can add new books, delete existing books, add new users, delete existing users and also edit the entries in either of the two databases. The main home page also has a “Book of the Day” section, which lists the book that is the most read by users across all databases. The user is also able to edit their username on the login dashboard and personalize their experience by borrowing books. Upon finishing whatever they want to do, the user has an option to logout.

Link to the Deployed Website:

Link: http://rfawesome21.pythonanywhere.com/

Analysis of the Web Application:

The developed website uses Flask-WTF forms to get the data from the users. The forms provide CSRF protection from third-party hackers and help in data validation. The forms are also, mainly objects of classes, so all the fields of a form come from a class.

The application uses 3 models- the user model, the roles model and the books model. Two many-to-many relations have been established- one between the role model and the user model – so every user can have multiple roles, and vice versa. The roles define the user as a user of the application or an admin. The other many-to-many relation that has been established is between the user and the book models, with the same concept as above.

My web application also uses sessions when a user is logged in. Whenever a user is logged in, the session stores the name, the ID and the email of the user. Whenever a user borrows a book, the session stores the name of the book. When the user logs out, all the data in the sessions is cleared.

My web application also uses flask-security for authentication. If a user is not registered, it does not let them log in. If the user enters a wrong password or email, they are not allowed to log in either. Upon clicking on the login button, the users are redirected to the login dashboard. The registration page also does not allow two users to have the same Email ID. Upon registering, the users are once again redirected to the login dashboard.

Logged in users can borrow and return books, and each book is added to their personal library which they can view at any time. The users can personalize their dashboard as they have the option of changing their username.

The application uses Bootstrap4, CSS and jQuery to appropriately style the application. The application is responsive on both tablet and computer devices and the design looks good when the window is resized.

The application used the logging library of flask. All the logs are stored in different levels; whenever a user navigates to another page, logs are stored at the ‘info level’ in the errorlogs.txt file. If an entry fails to get added to the database, the logs are stored at the ‘error level’ in errorlogs.txt and the error log is shows in the file.

The application uses the ‘aria-labels’ attribute in the html tags to make it accessible to the disabled. The application is deployed using PythonAnywhere.

Evaluation:

The application has been designed with heavy consideration to the user experience. All the features of the website are accessible and have been color coded, so they are clearly visible to the users.

The main page of the website clearly defines the purpose for which it should be used. Every webpage is loaded with moderate to high information to not burden the users with too much data to look at, at once. The background image is selected for every webpage on the application ensuring that it does not affect the rest of the content on the website which continues to remain clearly visible.

The design for every web page is unique to make sure that the users are not just looking at the same design over and over which makes for poor user experience. Uniformity is maintained throughout the website – it is ensured that the font and color for some features – like the title and the description of the books, the submit buttons throughout the website remain the same to avoid confusion among the users. Google fonts were also used to import new fonts and make the fonts more attractive. Google icons were used for displaying icons instead of even more text to make the content on the page more attractive to look at with aria-labels also implemented throughout to make sure these icons can even be accessible to the disabled who use screen readers.

The colors used on the website all matched with one another. Bootstrap4 and jQuery were also used in this website to make it responsive and accessible for every screen size. The navigation bar was stationed at the top of the page to make sure the links were clearly visible. There were no dead-end pages - every link clicked on the website led to another page. The users can easily borrow a book by just following the different links and special consideration was given to make sure this process was not complex at all. Upon clicking on a link, there was no delay in loading the next page – this was a very quick process and it ensured that the user didn't have to wait for long times because the server was busy. A special hamburger menu was designed as the navigation bar for smaller devices like mobiles and tablets.

If a user entered their wrong credentials while logging in or used an email which already existed somewhere in the database, the error messages displayed were concise and clear informing the user of their mistakes. The search field on the navigation bar was easy to find and the users could just enter part of the title of the book they wished to borrow and all the books with the search keywords were displayed on another webpage.

Security Issues:

Potential threats to the security of the application included attacks from third party CSRF attacks, broken authentication, sensitive data exposure and exploitation of the vulnerable components. Using WTF Forms minimized the risk of CSRF attacks. In order to prevent CSRF attacks, a token is sent from the server when the form is rendered and if that token is not sent back to the server from the form when it is submitted, a 422 (unprocessable entity) error will be thrown.[1]

The authentication used on the server is implemented using the flask security plugin. This plugin automatically handles logging in the user for the application, and if a user tries to login with invalid details, Flask-Security throws an error message and does not let the user login. Every time the user enters an email and a password, Flask-Security cross references it with the entry in the database model and if it matches, it allows the user to log in.

Flask security has an additional feature, such that when the user enters a password while registering, it automatically hashes it. The Hash Salt is configured as a ‘sha256’ hash, and it is very difficult to extract the original version from the hashed version.

The secret key is randomly assigned and is almost impossible to guess for any third-party hacker. If someone gets access to our server, they can see this key and decrypt tokens on their own. Hence, it is important to make it as difficult to guess as possible.

Third party hackers may also try and access data from the user’s sessions. For this reason, the sessions data in the application is hashed to sha-256 standards to prevent this from happening.