**Exercise 3**

**My first social media web application**

For our third assignment, the application we developed titled Squeak! is a simple social media application that allows users to register and log in and post short messages.

**Adding Authentication and Authorization**

Following the instructions of the assignment, we add the authentication and authorization flow where a user can either fill in the username and password and click on Sign Up for registering a new user whose credentials will be stored in the “passwd” file, or they can log in to the system by entering a valid combination of existing username and password and clicking on Sign In. The individual routes for these two events are **/signup** and **/signin** respectively, which are handled by the corresponding **postSignUp** and **postSignIn** methods in our controller.js file. Before this, the application main route **/** first checks the cookie to see whether a valid session is present, and if there already is a session present, then it automatically redirects to the **/home** route, otherwise the user is shown the combined Sign Up and Sign In page. This is handled in the **getIndex** method in the controller file.

Signin up a new user is handled in the **postSignUp** method in our controller, which also performs the following security checks as per our requirement:

1. The username is at least 4 characters long
2. The username is not already contained in the passwd file
3. The password is at least 8 characters long
4. The password does not contain the username

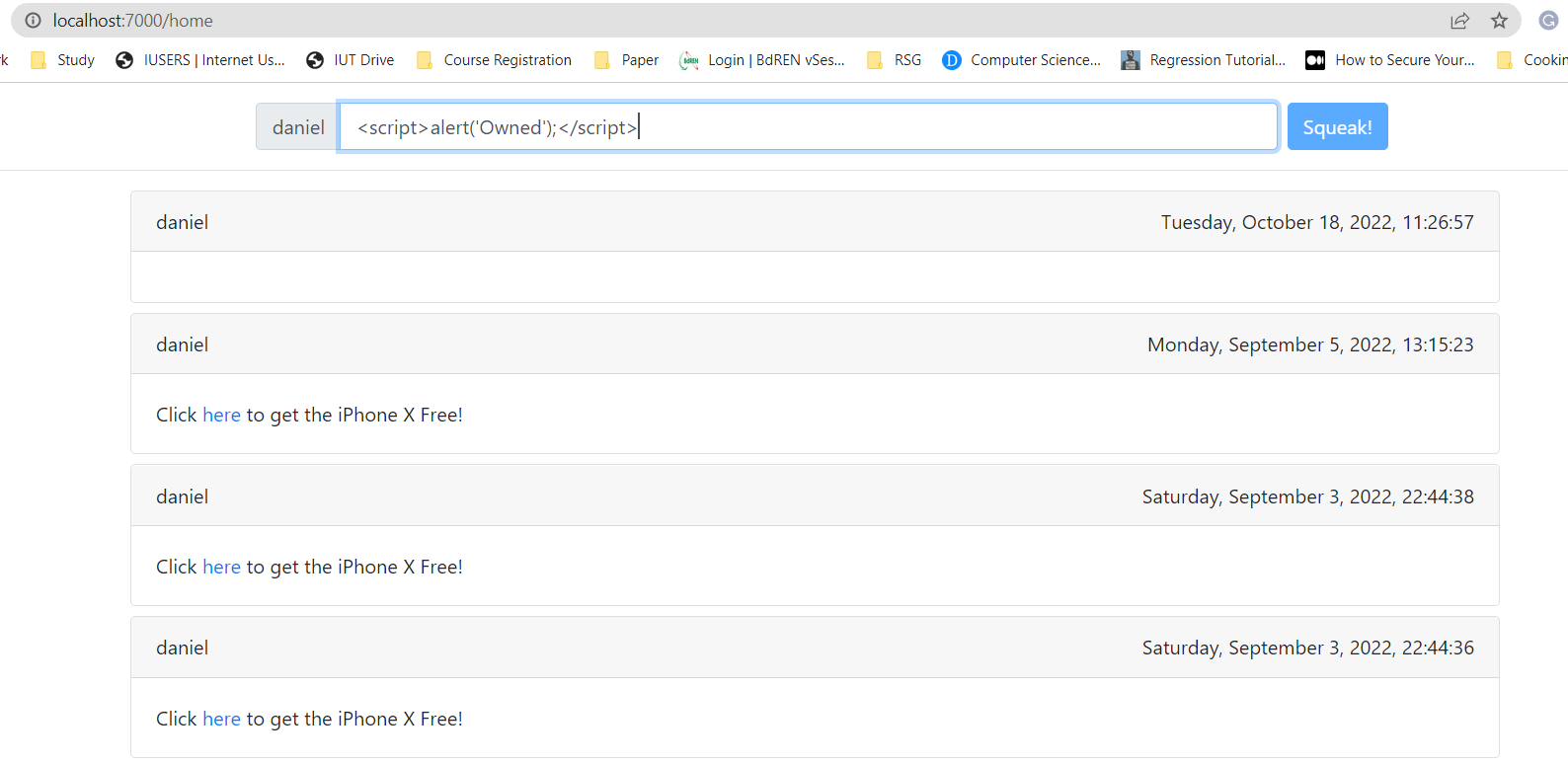
After registering a new user with valid username and password, the credentials get appended to the “passwd” file which holds the information in the following structure:

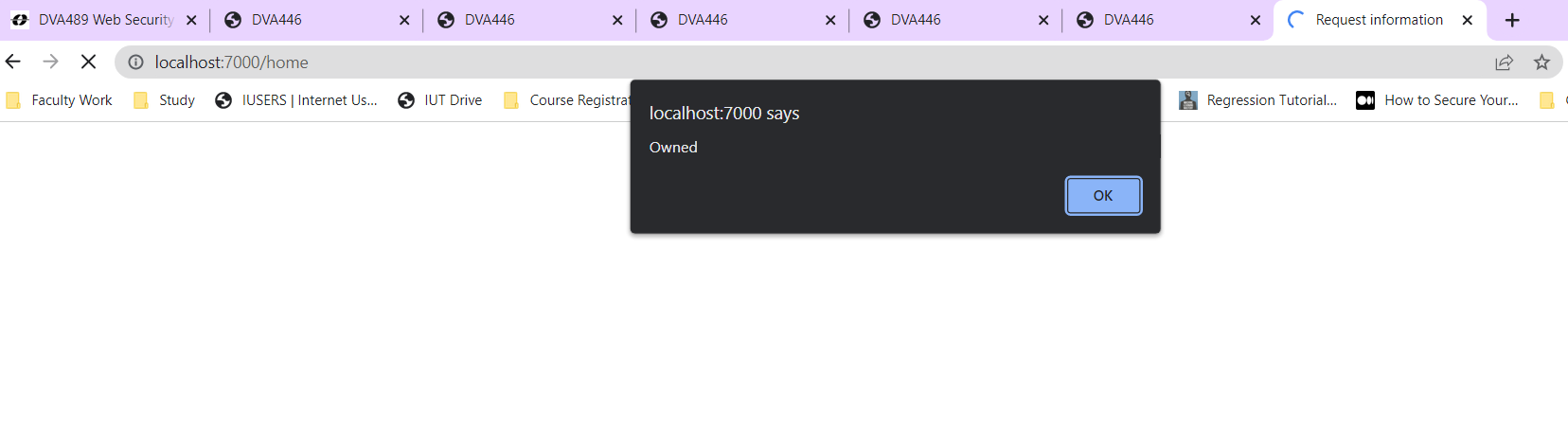
{"1":{"username":"daniel","password":"skaldjur"}}

The user is redirected to the Sign In page upon registering, where they can now log in using the credentials they used for signing up. Upon successful sign in, the user is presented with the home page where there is a textbox with a Squeak button at the top and the rest of the page contains the squeaks posted by all the users. This page is loaded by the **getHome** method in our controller, which fetches the squeaks from the “squeaks” file. Whenever a signed in user posts a new squeak, it gets appended to the “squeaks” file with the username, post content and the time by the **postHome** method in our controller. Finally, if the user clicks on the Sign Out button, then we redirect the user to the **/signout** route which calls the **getSignOut** method in our controller. This method clears the cookie and logs the user out by redirecting to the landing page by the **/** route.

**Stored XSS**

The Unpatched version of our application is hosted in **localhost:7000** and it was deliberately made to have no protection against the Cross Site Scripting attack because there was no mechanism added for filtering or escaping special characters in the post. So we can write a script in the post and it would execute the script as shown in the following two images:





Also, it is possible to steal the cookie since we are intentionally not using the “httpOnly” and “secure” attributes.

**Protecting the application**

There are two steps to protecting our application from Stored XSS attacks. First is to escape all untrusted data in the posts. Next is to protect the cookie by setting its attributes. We implement these measures in the **Patched version** of our application, which is hosted in **localhost:8000**.

Instead of using our own escaping mechanism, we reimplement our application using Mustache which automatically escapes all data when generating pages. We reuse code from our html files and create new files under the “views” directory naming them as “landingPageView.mustache” and “homeView.mustache”.

We also protect our cookie by setting the “httpOnly” and “secure” attributes to true and also adding a “maxAge” attribute when setting the cookie upon successful sign in. This part is handled in the **postSignIn** method in our controller. Now, it is not possible to mount the attack as we can see in the image below:

