Introduction

This project is a lottery website. A user may register an account and login. Upon doing so, the user may pick a set of 6 different numbers between 0 and 60. The user may add any number of sets, and check if their numbers have won them the lottery. A user may also register as an admin

Approach

Data Input – General overview

A general overview of data input: upon an attempt of submission client side checks written in JavaScript occur. If they pass, server side checks occur using servlet filter. Then finally the form is submitted in which the appropriate servlet is called and executed.

Data Input – Client Side Checks

Currently client-side checks ensure the phone number contains only numbers using regular expression. Also the format is validated by checking the number of characters inputted and whether there are dashes at specified positions. The password is also ensured to be between 8 and 15 characters by examining and comparing the length. Furthermore the password is checked for at least one digit, upper and lowercase character using regular expression. If any of these checks fail the form is never submitted & server side checks never occur. Nor is the form submitted. Instead the user is alerted with a message describing their error.

Data Input – Server Side Checks

If however the user passes client side checks, server side checks begin. The form the user has submitted sends a request. However the request is intercepted by a mapped servlet filter. Server side checks for illegal input that could possibly result in SQL injections. The value from each user input field is checked for whether or not it contains any illegal string. If illegal input is not detected the ‘chain’ continues as normal. Where the ‘chain’ refers to the regular sequence of requests and responses. However if illegal input is detected the chain is not continued as normal instead the user is redirected to an error page.

Problem Solving

My objective was to validate user input. I chose JavaScript over JQuery for its simplicity and stronger similarity to Java. Through observing the behaviour of interaction with the webpages. I learned that Upon submission the corresponding servlet written in Java is executed. From research I learned one may embed a JavaScript function within the html file of the webpage. Furthermore the html form itself can contain an attribute ‘onsubmit’ which contains a function. If the contained function returns false the form is not submitted otherwise the form is submitted. Lastly I researched acquiring the form input from accessing elements by ID. From researching these specific topics I was able to implement a solution. Specific user input is acquired, on submission a function performs basic sequential string comparisons,

My objective was to prevent SQL injections via preventing specified disallowed input. If illegal input was detected the user should be redirected to an error page.

Firstly the data from the user is inputted through a form written in html. The database contains a table, ‘userAccounts’ in which every column has a limit of 50 characters. If the

data from the submitted form exceeds this limit the database fails. To prevent this case, each input field contains the tag ‘maxlength=50’ preventing input from exceeding the limit.

Testing

To test my final product I followed the instructions to execute the project using docker only from the command line. This was to ensure all elements of the program function correctly when shipped (submitted) under the same execution conditions.

Testing Input

My testing strategy for input in html fields consisted of inputting valid values, erroneous values, edge values, and invalid values where possible. This was to ensure fields for registering an account accepted the input stated in the requirement. But also rejected any input that did not meet the specified requirement. This also included inputting the keywords prevented by the ServletFilter to prevent SQL injections. To again, ensure the servlet filter was performing as expected.

Testing Hash Creation & Storage

To ensure a hash was being created from an imported function I displayed the return value. From observation I was able to confirm a hash can be created. Furthermore a requirement was to use the user’s hashed password for login verification. To ensure this was possible, two identical strings of length 10 were hashed and compared to determine whether the hashes were equal. The resultant hashes were equal, which ensured the hashing algorithm could be used for login verification.
To ensure the hash was being stored in the database as opposed to the password. I confirmed from observation of the database the password columns contained hashes. As all strings under that column began with ‘$2a$10’ which is apart of the hashing algorithm used.

Testing User Roles (Admin & Non-admin)

To ensure non-admin and admin users were directed to their corresponding webpage stated in requirements. I created separate accounts for the different roles and upon logging in verified that the correct pages were displayed. To ensure neither role had access to their counterpart’s webpage I removed any links between the two pages.

Testing Decryption, Encryption, Public & Private Keys, & Input/Output with Files

Numerous strings were encrypted, written to a file, and read into an array. Each string in the array was then decrypted and outputted expected results. Which also ensured the public and private keys were being used correctly. To reduce waiting time and to simplify the testing process, I first ran these tests in a separate class. To also ensure that the user logged in, could only modify their own numbers. I logged into separate accounts to add and view numbers. Both accounts yielded different tables of sets. Ensuring the encryption and decryption requirements were met.

Recommendations

Reflection