

WAPH-Web Application Programming and Hacking

Instructor : Dr Phu Phung

Student:

Name: Ruthvik Suvarnakanti

Uc-mail: suvarnrk@mail.uc.edu



Individual Project 2: Full-stack Web Application Development

Git Repository: <https://github.com/suvarnrk/waph-suvarnrk/tree/main/project2>

Overview: The functional requirement of the project includes creating pages like Registration page, Login page, View and Edit user profile pages and Changing password page. The user registration process involves checking data both on the user's device and on the server to make sure everything is correct and secure. This includes making sure usernames and email addresses haven't been used before. When users log in, their passwords are encrypted for safety, and the system remembers who they are as they move around the site. Users can see and change their profile details like name and email, and when they update their password, the system makes sure it's strong enough. To keep everything safe, the system also uses encryption, protects against certain types of attacks, limits how many login attempts can be made, keeps a record of important events, and gives clear feedback to users about what's happening. There are additional requirements like the application needs to be careful about security. It should use HTTPS for deployment, hash passwords before saving them, not use the MySQL root account directly in PHP code, and always use prepared statements for database operations to avoid SQL injection attacks. Input validation should be thorough on both the client and server sides to stop things like XSS attacks. User data should be stored safely in a MySQL database, with everything properly organized and encrypted when needed. When building the front-end with HTML, CSS, and JavaScript, it's important to validate user input on the client side. Also, the application needs to manage user sessions securely and protect against CSRF attacks, like using anti-CSRF tokens. Taking care of these things will make sure the application keeps user data safe from hackers and other threats. All these have been covered during the development and execution.

video demonstration: <https://youtu.be/ARly22r1B0g>

Part 1: Functional Requirements

1) User Registration:

The user registration system lets people sign up by providing their username, password, name, and email. It makes sure all the information entered is correct and secure, both while users are typing it in and when it's sent to the server. If anything doesn't meet the requirements, like a missing field or an invalid email, it lets users know so they can fix it. Once everything looks good, the system saves the details in the database. If there are any problems along the way, it tells users what's wrong and helps them through it. The goal is to make signing up easy and safe for everyone involved. Using HTTPS for sending data to the backend adds extra protection, keeping user details safe while they're being transferred. The backend processes the information securely, checking it and saving it in the database for later use. The webpage looks nice thanks to HTML and CSS, making it more enjoyable for users.

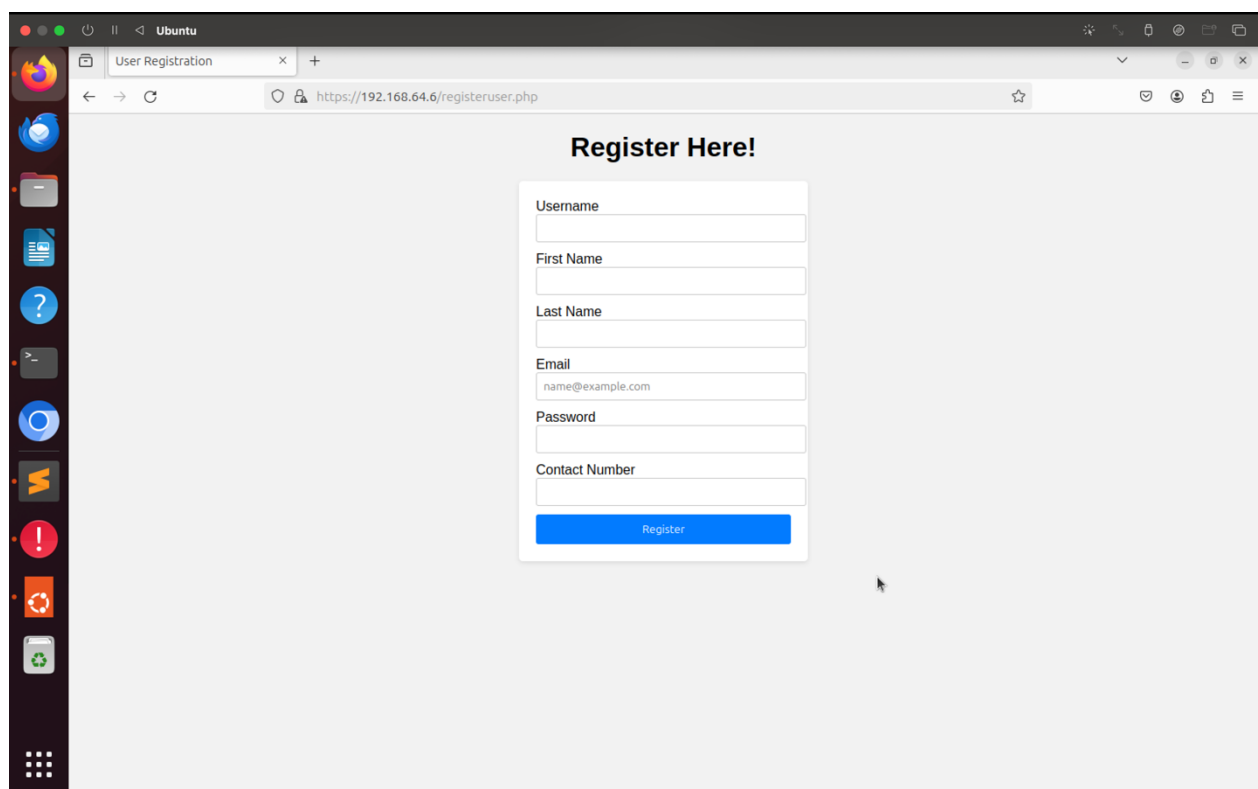
A screenshot of a web browser window on an Ubuntu desktop. The browser's address bar shows the URL 'https://192.168.64.6/registeruser.php'. The page has a light gray background and a central white registration form. The form is titled 'Register Here!' in bold black text. It contains several input fields: 'Username', 'First Name', 'Last Name', 'Email' (with a placeholder 'name@example.com'), 'Password', and 'Contact Number'. Below these fields is a blue button labeled 'Register'. The browser's sidebar on the left shows various application icons, and the top of the window displays the Ubuntu logo and window controls.

Fig 1: Registering New User form

Code for new user registering form:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
```

```

<title>User Registration</title>
<link rel="stylesheet" type="text/css" href="style.css">
<script type="text/javascript">
    function displayTime() {
        document.getElementById('digit-clock').innerHTML = "Current time: " + new Date();
    }
    setInterval(displayTime, 500);

    function validateForm() {
        var email = document.forms["registrationForm"]["email"].value;
        var phone = document.forms["registrationForm"]["phonenumber"].value;
        var password = document.forms["registrationForm"]["password"].value;

        // Email validation
        var emailPattern = /^[\\w.-]+@[\\w.-]+(\\.([\\w.-]+))*$/;
        if (!emailPattern.test(email)) {
            alert("Please provide a valid email address");
            return false;
        }

        // Phone number validation
        var phonePattern = /^\\d{10}$/;
        if (!phonePattern.test(phone)) {
            alert("Mobile number must be a 10-digit");
            return false;
        }

        // Password validation
        var passwordPattern = /^(?=.*[A-Za-z])(?=.*\\d)[A-Za-z\\d]{8,}$/;
        if (!passwordPattern.test(password)) {
            alert("Password must contain at least one letter, one number, and be at least 8 characters long");
            return false;
        }
    }
</script>
</head>
<body>
    <h1>Register Here!</h1>

    <form name="registrationForm" action="newuser.php" method="POST" class="form login"
onsubmit="return validateForm();">
        Username <input type="text" class="text_field" name="username" required> <br>
        First Name <input type="text" class="text_field" name="firstname" required><br>
        Last Name<input type="text" class="text_field" name="lastname" required><br>
        Email<input type="email" class="text_field" name="email" required
placeholder="name@example.com"><br>
        Password <input type="password" class="text_field" name="password" required><br>
        Contact Number <input type="tel" class="text_field" name="phonenumber" required pattern="[0-
9]{10}" title="Please enter a 10-digit phone number"><br>
        <button class="button" type="submit">Register</button>
    </form>

```

</body>

</html>

2) Login: In my secure login setup, users enter their username and password to log in, and if these are correct, they get access to their profiles. To keep track of users as they move around the site, session management is used, which means creating session cookies once users log in. These cookies have settings to make sure they're secure, like only being sent over encrypted connections and not being accessible to scripts on the user's device. During a user's session, checks are made to ensure everything stays safe, like logging them out if they're inactive for too long or if they switch browsers. HTTPS is used to encrypt data as it travels between users and the server, and there are systems in place to handle any errors and keep an eye out for anything suspicious. Also, passwords are stored in a secure way to protect them from being stolen. Overall, the system is focused on keeping users' accounts safe and making sure the login process is smooth and secure.

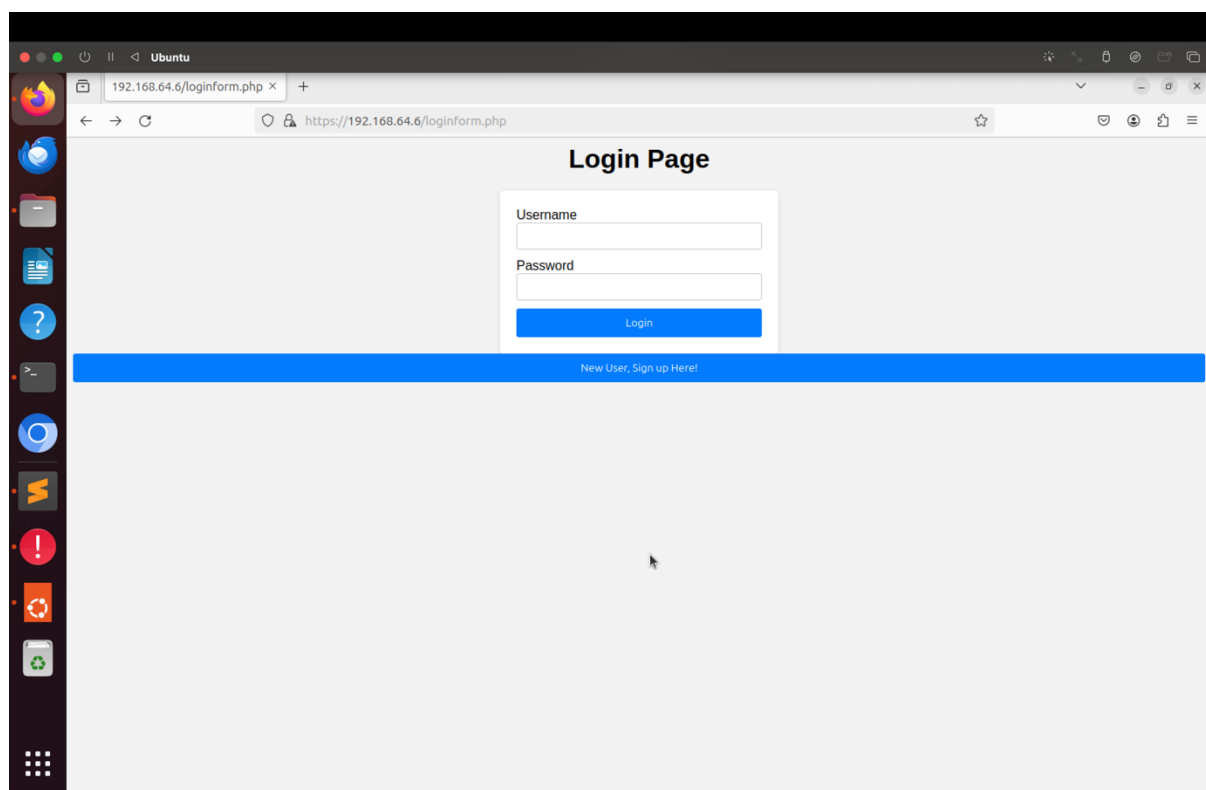


Fig2: Login Page

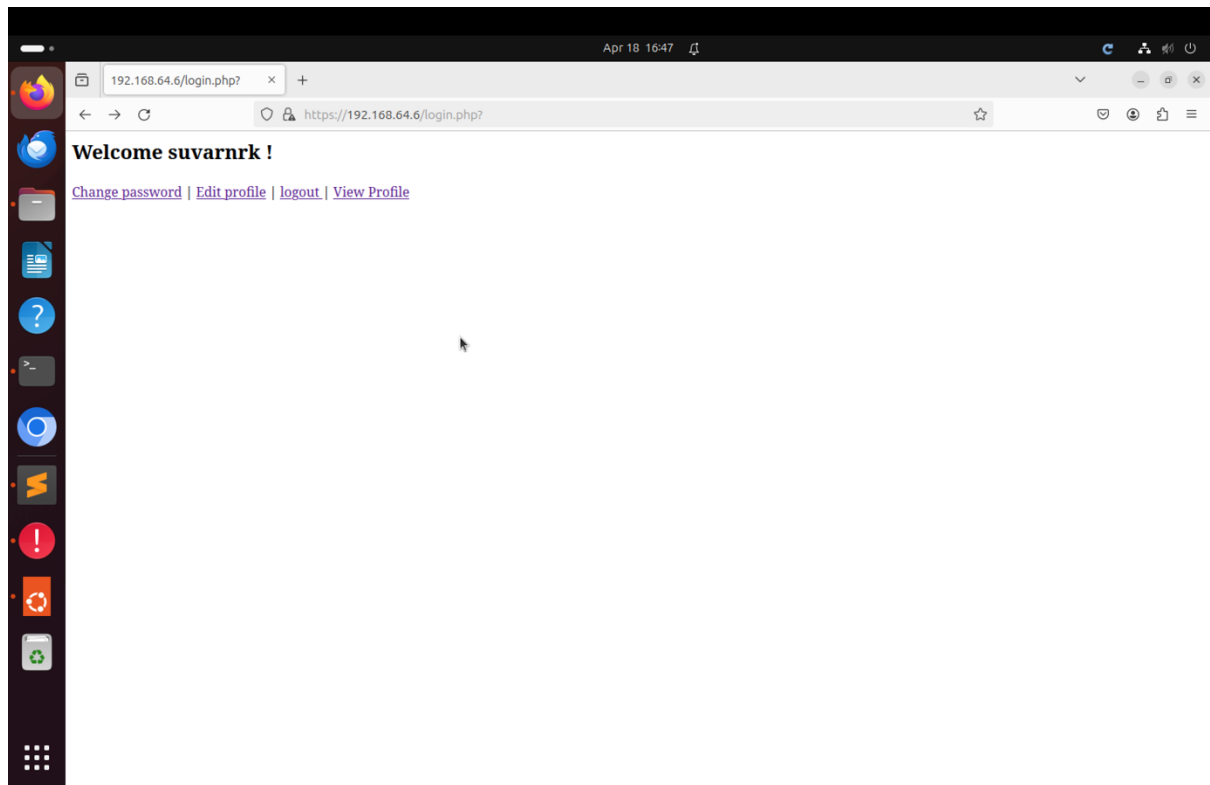


Fig3: Dashboard after successful authentication

3) Profile Management: Profile management lets users see and update their profile details like name and email. Once logged in, users can view their profile to check their information, and they can also edit it if needed. Only users who are logged in can access and change their profiles, and the system checks that any changes are valid and safe. Users get feedback to confirm changes and help with any mistakes. Some systems also let users change their profile picture, with everything logged for safety. Overall, profile management gives users more control over their accounts while making sure everything stays secure and accurate.

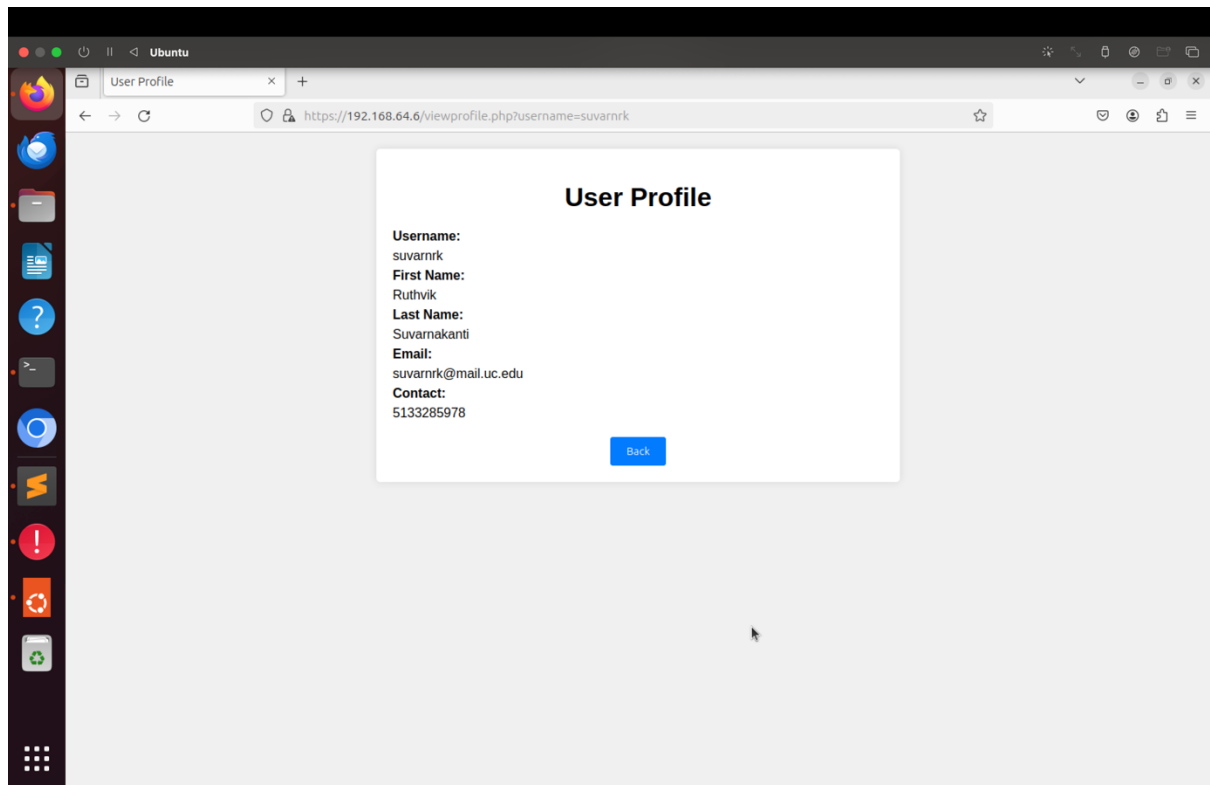


Fig4: User Details after successful login

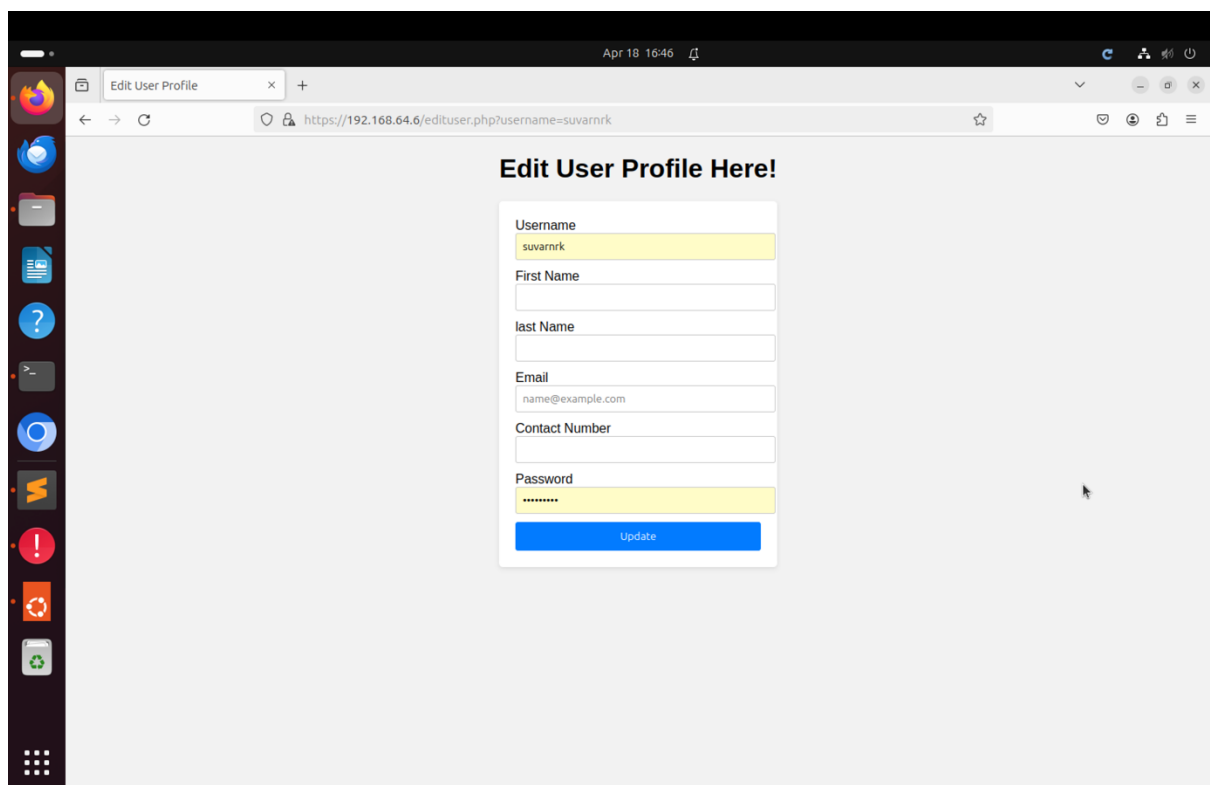


Fig5: Edit User details after logged In.

4) Password Change: The password update feature allows users to change their passwords securely. First, users log in to confirm their identity. Then, they can access a form where they enter their current password and set a new one. The system checks that the new password meets security standards and encrypts all data transmission. After the change, users receive confirmation, and any errors are explained clearly. Additional security measures like password history and expiration policies can be added if needed. Lastly, all password changes are logged for monitoring and auditing purposes to ensure user accounts stay safe.

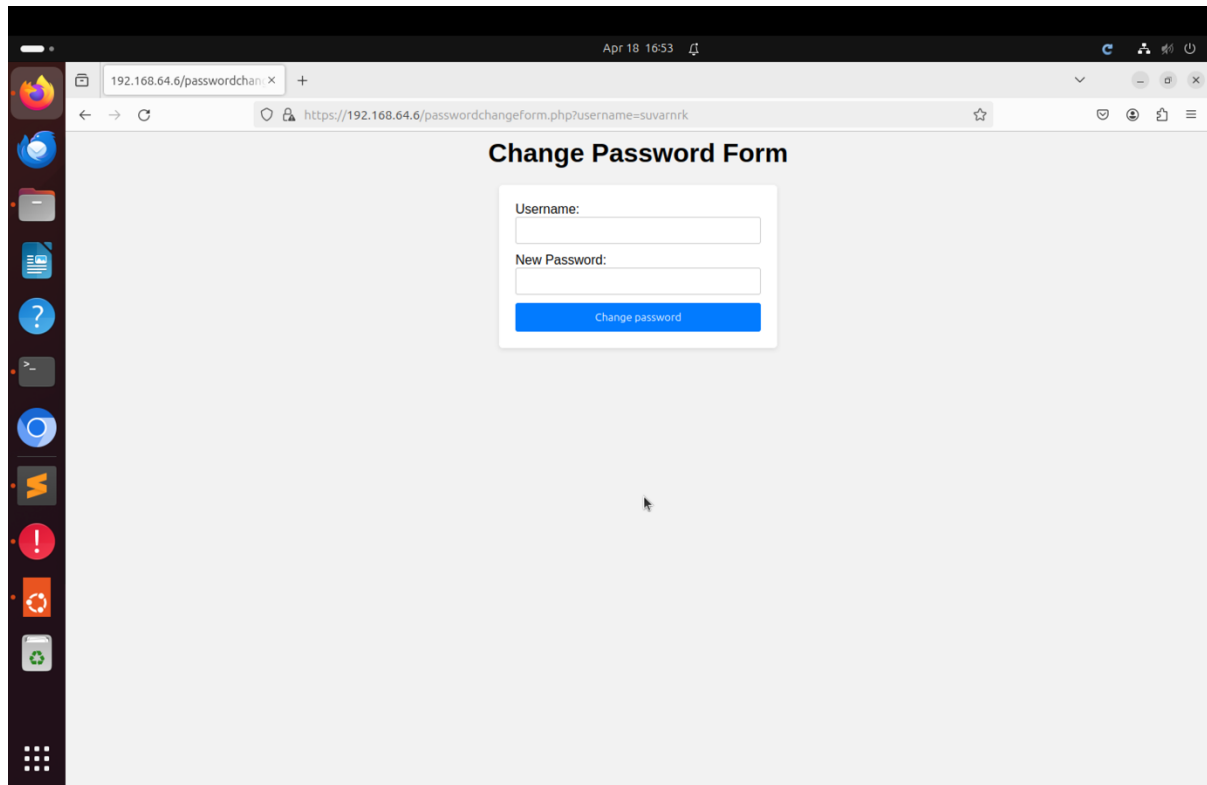


Fig 6: Change Password Form

Security and Non-technical Requirements

1) Security: To keep the application secure, I implemented few important steps to take. First, it's served over HTTPS to protect data sent between users and the server. Then, before storing any passwords, use a strong hashing method like crypt to scramble them. Avoided using the MySQL root account directly in PHP code; instead, create a separate user with limited access rights. Lastly, whenever the application interacts with the database, used prepared statements to prevent sneaky attacks that could mess with the data. These precautions helped and ensured user information stays safe and the application stays secure.

```
suvarnrk@ubuntu: ~/waph-suvarnrk/project2
suvarnrk@ubuntu:~/waph-suvarnrk/project2$ sudo mysql -u suvarnrk -p
[sudo] password for suvarnrk:
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 289
Server version: 8.0.36-0ubuntu0.23.10.1 (Ubuntu)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use waph
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_waph |
+-----+
| users           |
+-----+
1 row in set (0.02 sec)

mysql> select * from users;
+-----+-----+-----+-----+-----+-----+
| username | firstname | lastname | email | phonenumber | password |
+-----+-----+-----+-----+-----+-----+
| admin    | max       | karl     | max@gmail.com | 3281784567 | pass     |
| tata     | tata     | birla    | tata@gmail.com | 9848212345 | password123 |
| tommy    | tommy     | hilf     | tommy@gmail.com | 9182445524 | d00653e37e58e7cb7e2ad6902ef677c4 |
| suvarnrk | Ruthvik   | Suvarnakanti | suvarnrk@mail.uc.edu | 5133285978 | 7c6a180b36896a0a8c02787eeafb0e4c |
| Ramsy    | Ram       | Syleka   | ramsy@gmail.com | 5134567898 | 482c811da5d5b4bc6d497ffa98491e38 |
| kennys   | Kenny     | barth    | kennys@gmail.com | 5134567801 | 482c811da5d5b4bc6d497ffa98491e38 |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

Fig7: Hashed passwords in the user's table

2) Input Validation: To prevent common web vulnerabilities like Cross-Site Scripting (XSS) attacks, it's crucial to thoroughly validate user input on both the client and server sides and for this I have implemented validation methods. On the client side, JavaScript validation ensures that user inputs meet expected criteria before submission, providing immediate feedback to users if errors are found. On the server side, input validation checks data received from the client to ensure it's safe and meets required standards. Additionally, encoding user-generated content and implementing strict Content Security Policy (CSP) headers helps mitigate the risk of XSS attacks by preventing malicious code execution in browsers. Regular updates and security audits further strengthen the input validation process, ensuring ongoing protection against evolving threats. By implementing these measures, web applications can maintain security and integrity, safeguarding against XSS and other vulnerabilities.


```
waph-suvarnrk / project2 / registeruser.php ↑ Top
Code Blame 54 lines (49 loc) · 2.11 KB Code 55% faster with GitHub Copilot Raw ⌵ ⌵ ⌵ ⌵ ⌵
10     }
11     setInterval(displayTime, 500);
12
13     function validateForm() {
14         var email = document.forms["registrationForm"]["email"].value;
15         var phone = document.forms["registrationForm"]["phonenumber"].value;
16         var password = document.forms["registrationForm"]["password"].value;
17
18         // Email validation
19         var emailPattern = /^[w.-]+@[w-]+\.[w-]+$/;
20         if (!emailPattern.test(email)) {
21             alert("Please provide a valid email address");
22             return false;
23         }
24
25         // Phone number validation
26         var phonePattern = /^d{10}$/;
27         if (!phonePattern.test(phone)) {
28             alert("Mobile number must be a 10-digit");
29             return false;
30         }
31
32         // Password validation
33         var passwordPattern = /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$/;
34         if (!passwordPattern.test(password)) {
35             alert("Password must contain at least one letter, one number, and be at least 8 characters long");
36             return false;
37         }
38     }
39 </script>
40 </head>
41 <body>
42 <h1>Register Here!</h1>
43
```

Fig8: Client-side validations on registering new user page

3) Database Design : To ensure user information remains safe in a MySQL database, I created a structured layout for storing data, focusing on tables for user accounts while keeping data organized and efficient. Encrypt passwords before saving them to prevent unauthorized access and use secure methods like parameterized queries to interact with the database, thwarting attempts at SQL injection attacks. Tighten access controls and permissions, and regularly update database software to tackle any vulnerabilities. Keep regular backups, stored securely, and employ monitoring and auditing tools to promptly catch and respond to any suspicious activities. These steps work together to bolster the security of the MySQL database, safeguarding user details from potential threats.

```

1
2 <?php
3 $mysql= new mysqli('localhost','suvarnrk','ubuntu','waph');
4 if($mysql->connect_errno){
5     printf("Database connection failed: %s\n", $mysql->connect_error);
6     return FALSE;
7 }
8 function newuser($username,$firstname,$lastname,$email,$phonenumber,$password){
9     global $mysql;
10    $prepared_sql = "INSERT INTO users(username,firstname,lastname,email,phonenumber,password) VALUES(?,?,?.?,?.?,md5(?));";
11    $stmt = $mysql->prepare($prepared_sql);
12    $stmt->bind_param("ssssss", $username,$firstname,$lastname,$email,$phonenumber,$password);
13    if($stmt->execute())return TRUE;
14    return FALSE;
15 }
16 function userupdate($username,$firstname,$lastname,$email,$phonenumber,$password){
17     global $mysql;
18     $prepared_sql = "UPDATE users SET firstname=?,lastname=?, email=?, phonenumber=?,password=md5(?) WHERE username=?";
19     $stmt = $mysql->prepare($prepared_sql);
20     $stmt->bind_param("ssssss", $username,$firstname,$lastname,$email,$phonenumber,$password);
21     if($stmt->execute())return TRUE;
22     return FALSE;
23 }
24
25 function userprofileview($username)
26 {
27     global $mysql;
28     $prepared_sql = "SELECT * FROM users WHERE username = ?";
29     $stmt = $mysql->prepare($prepared_sql);
30     $stmt->bind_param("s", $username);
31     $stmt->execute();
32     $result = $stmt->get_result();
33     if ($result->num_rows == 1) {
34         return $result->fetch_assoc();
35     }
36     return null;
37 }
38 function checklogin($username,$password) {
39     global $mysql;
40     $prepared_sql = "SELECT * FROM users WHERE username= ? AND password = MD5(?)";
41     $stmt = $mysql->prepare($prepared_sql);
42     $stmt->bind_param("ss",$username,$password);
43     $stmt->execute();
44     $result=$stmt->get_result();
45     if ($result->num_rows == 1) return TRUE;
46     return FALSE;
47 }
48 function passwordchange($username,$password) {
49     global $mysql;
50     $prepared_sql = "UPDATE users SET password = md5(?) WHERE username= ?";
51     $stmt = $mysql->prepare($prepared_sql);
52     $stmt->bind_param("ss",$password,$username);
53     if ($stmt->execute()) return TRUE;
54     return FALSE;
55 }
56
57
58

```

Fig9: SQL statements using database.php

4) Front-end Development: To create a user-friendly interface, HTML, CSS, and JavaScript are combined. HTML organizes content, CSS enhances visuals, and JavaScript adds interactivity. By using responsive design principles and client-side validations, the interface adapts to different devices and ensures data accuracy. Additionally, accessibility considerations ensure inclusivity for all users. Overall, this approach results in an intuitive interface that provides a pleasant browsing experience across various devices.

5) Session Management: To ensure secure user authentication, session management involved creating unique session identifiers after login, which are securely stored on the server. These session tokens should be transmitted over HTTPS to prevent interception. Measures like setting expiration times and tying sessions to specific IP addresses and User-Agent headers help defend against session hijacking and fixation. Monitoring session activity and logging events also play a crucial role in detecting and addressing suspicious behavior. Overall, these practices ensure user sessions remain secure and protected from unauthorized access.

6) CSRF Protection: To safeguard against CSRF attacks during database modifications, it's vital to use anti-CSRF tokens. These tokens, unique to each user session, are generated securely and verified during requests to confirm their validity. They're incorporated into form submissions or custom HTTP headers. By mandating and checking these tokens, unauthorized alterations are thwarted. It's important to extend this safeguard to sensitive operations and keep logs for monitoring and analysis. Overall, anti-CSRF tokens act as a strong defense, enhancing the security of database modification processes against CSRF attacks.