Candidate No: 215709

Introduction to Computer Security

Coursework 1

Report

1) Application URL: http://139.162.217.208

2) Code file Location: https://1drv.ms/u/s!AjapXz_0gKL1geFv_9puJpVRMUaydw?e=7gmXgv

3) Test users' detail:

Admin:

- Username: admin
- Password: Admin1234admin

Standard user:

- Username: user
- Password: User1234user

4) Recording: There is a recording of me showing off the password reset feature using an email address inside the code file location. This will not be possible with the above accounts as you cannot access the email address.

Task 0 – Self-reflection

Candidate No: 215709

Excellent (10-9 marks) **Student must have gone beyond**	**Good (8-6 marks)**	**Average (5-3 marks)**	**Poor (2-0 marks)**	**Criteria**
Policy has no flaw, and its implementation is excellent. Various mechanisms implemented to ensure password policy is secure.	Policy has no flaws, but implementation of policy is simple.	Password policy has very few flaws. However, different sections of policy are implemented and working.	Policy has many flaws for example password is not encrypted, and no salt applied. Password forgot policy has security flaws.	**Password policy 10marks** Password entropy, encrypted storage, security questions and recovery of password
Several countermeasures are implemented, and the quality of countermeasures are excellent.	Countermeasures are implemented in all the pages however quality of implementation is simple.	Implemented countermeasures only in some parts of the application. +	Very little effort to implement countermeasures to avoid these vulnerabilities.	**Vulnerabilities 10 marks** SQL injection, XSS, CSRF, File Upload and any other obvious vulnerability. SQL injection prevented, file upload only allows permitted types, CSRF possibly missing.
All the requirements are implemented to authenticate users. Implementation quality is excellent.	All requirements are implemented to authenticate the user. However, quality of implementation is simple.	Only some obvious requirements are not implemented.	Lots of obvious authentication's requirements are not implemented.	**Authentication 10 marks** User identity management (registration and login etc), Email verification for registration, 2 factor authentications (PIN and or email), Only 2 factor authentication is not completed, verification email is almost done but is flawed.
Excellent implementation of countermeasures against these attacks.	No flaws in countermeasures however quality of implementation is simple.	Some flaws in countermeasures	Very little effort against these attacks.	**Obfuscation/Common attacks 10 marks** Brute force attack – Number of attempts Botnet attack – Captcha Dictionary attack/Rainbow table attack The password hashing function in PHP helps prevent against dictionary / rainbow table attacks because of the salt.
Implementation of other security features has no flaws. No obvious security feature is ignored.	Several security features implemented. Implementation has flaws.	Other security features are implemented but obvious ones are ignored.	Very little effort to implement some obvious other security features like storage of confidential information.	**Other security features like confidentiality of important information 10 marks** For example, identify information that needs to be stored as encrypted. Not attempted
Claimed features are complex. Quality of achievement is excellent.	Claimed features are complex however quality of achievement/implementation could have been better.	Claimed features are somewhat complex and implementation could have been better.	Claimed features are not complex and challenging.	**Deeper understanding, two extra web security 10 marks** Carry out your investigation and implement two more security features. These need to be complex and challenging one. User can delete their own data at any point from their account

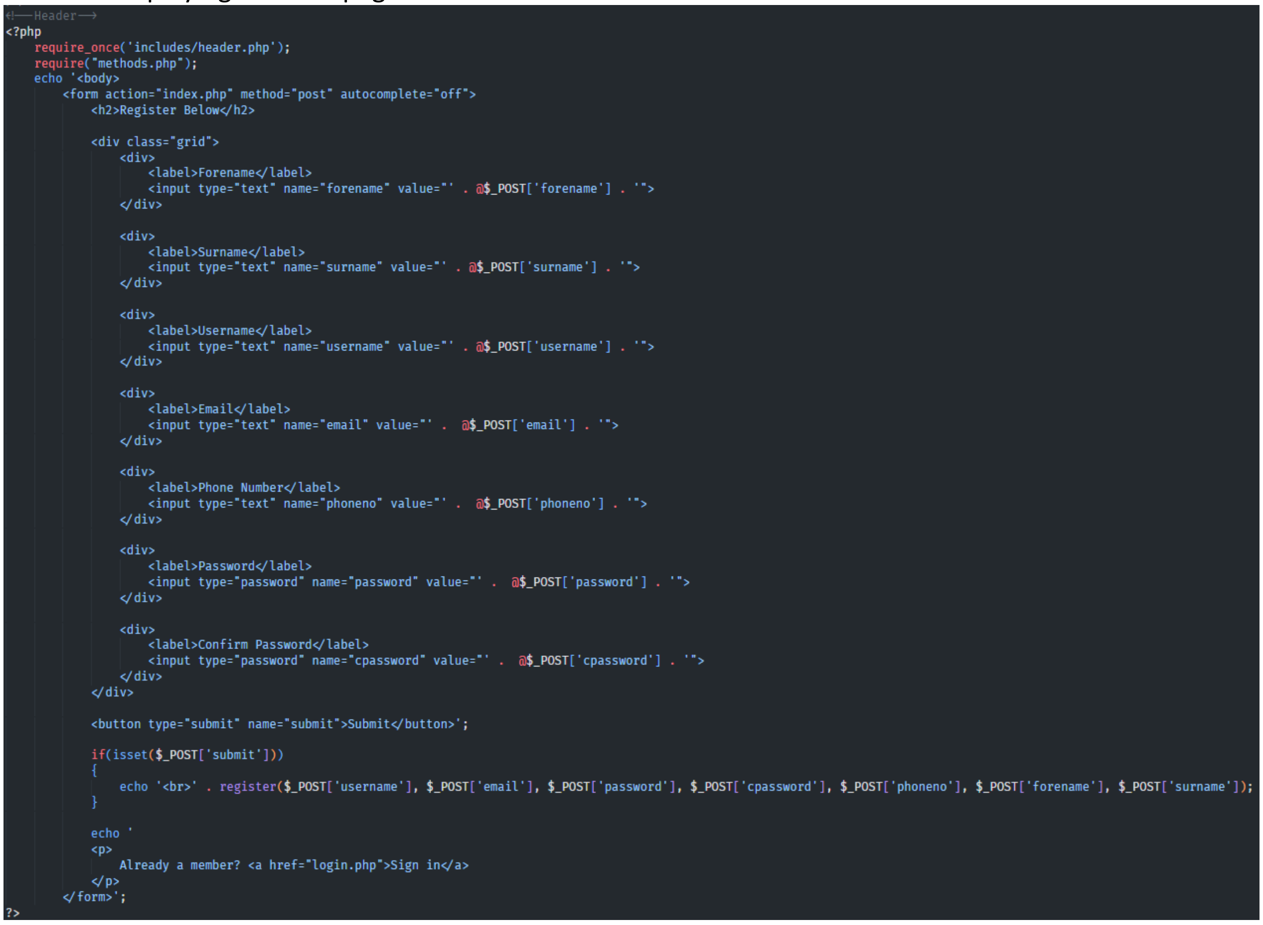
5 marks	**5 marks**	**5 marks**	**5 marks**	**5 marks**	**10 marks**	
List evaluation-Task6	Request evaluation – task 5	Request evaluation – task 4	Forgot password-Task3	Login-Task2	User registration/Database-Task1	**Features of webs application**
Done	Done	Done	Done	Done	Done	

Candidate No: 215709

Up to 5 marks	**0 marks**	
Fully completed	Marking not completed	**Self-reflection**

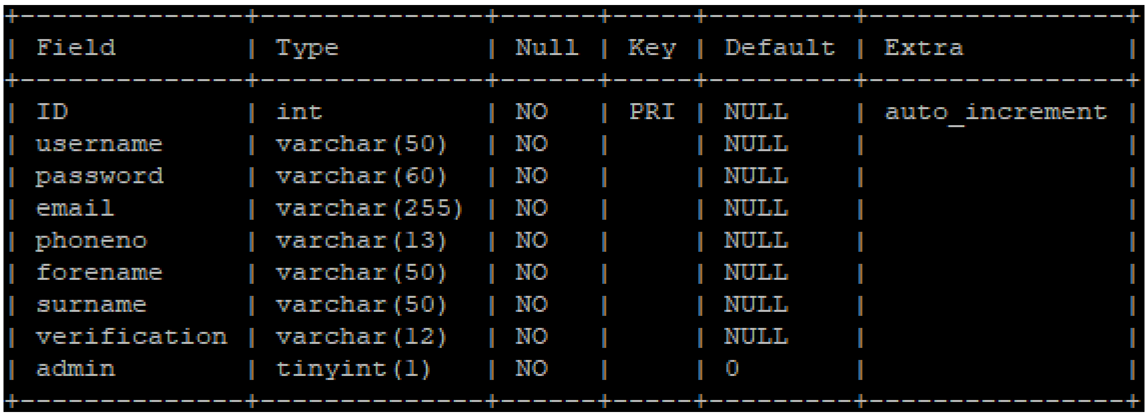
Task 1 – User registration

Code for displaying the web page to the user.

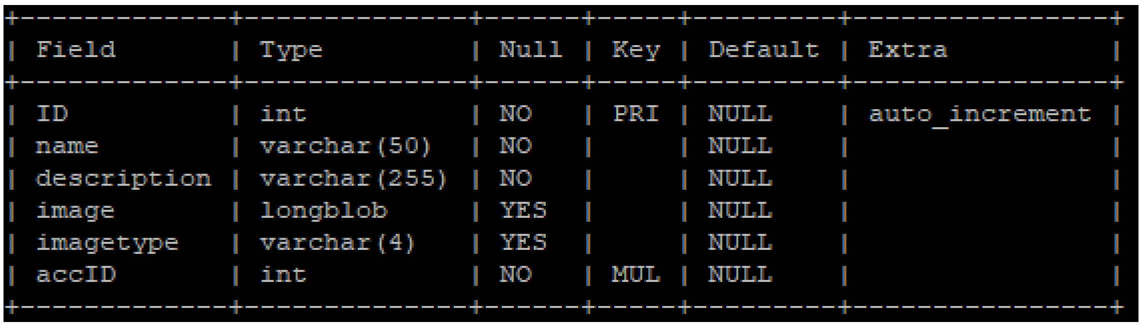


```
<!——Header——>
<?php
    require_once('includes/header.php');
    require("methods.php");
    echo '<body>
        <form action="index.php" method="post" autocomplete="off">
            <h2>Register Below</h2>

            <div class="grid">
                <div>
                    <label>Forename</label>
                    <input type="text" name="forename" value="' . @$_POST['forename'] . '">
                </div>

                <div>
                    <label>Surname</label>
                    <input type="text" name="surname" value="' . @$_POST['surname'] . '">
                </div>

                <div>
                    <label>Username</label>
                    <input type="text" name="username" value="' . @$_POST['username'] . '">
                </div>

                <div>
                    <label>Email</label>
                    <input type="text" name="email" value="' .  @$_POST['email'] . '">
                </div>

                <div>
                    <label>Phone Number</label>
                    <input type="text" name="phoneno" value="' .  @$_POST['phoneno'] . '">
                </div>

                <div>
                    <label>Password</label>
                    <input type="password" name="password" value="' .  @$_POST['password'] . '">
                </div>

                <div>
                    <label>Confirm Password</label>
                    <input type="password" name="cpassword" value="' .  @$_POST['cpassword'] . '">
                </div>
            </div>

            <button type="submit" name="submit">Submit</button>';

            if(isset($_POST['submit']))
            {
                echo '<br>' . register($_POST['username'], $_POST['email'], $_POST['password'], $_POST['cpassword'], $_POST['phoneno'], $_POST['forename'], $_POST['surname']);
            }

            echo '
            <p>
                Already a member? <a href="login.php">Sign in</a>
            </p>
        </form>';
?>
```

Database Tables ('accounts' and 'evaluations' respectively



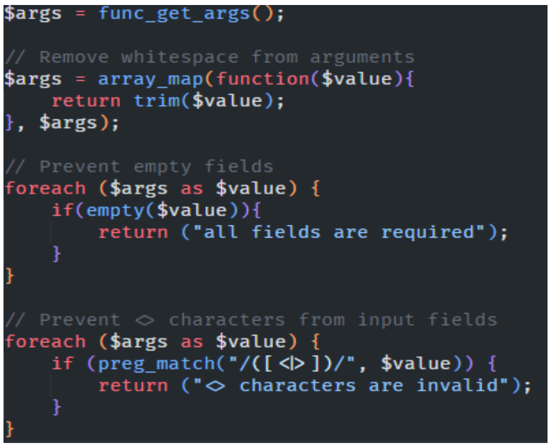
```
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| ID           | int          | NO   | PRI | NULL    | auto_increment |
| username     | varchar(50)  | NO   |     | NULL    |                |
| password     | varchar(60)  | NO   |     | NULL    |                |
| email        | varchar(255) | NO   |     | NULL    |                |
| phoneno      | varchar(13)  | NO   |     | NULL    |                |
| forename     | varchar(50)  | NO   |     | NULL    |                |
| surname      | varchar(50)  | NO   |     | NULL    |                |
| verification | varchar(12)  | NO   |     | NULL    |                |
| admin        | tinyint(1)   | NO   |     | 0       |                |
+--------------+--------------+------+-----+---------+----------------+
```



```
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| ID          | int          | NO   | PRI | NULL    | auto_increment |
| name        | varchar(50)  | NO   |     | NULL    |                |
| description | varchar(255) | NO   |     | NULL    |                |
| image       | longblob     | YES  |     | NULL    |                |
| imagetype   | varchar(4)   | YES  |     | NULL    |                |
| accID       | int          | NO   | MUL | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
```

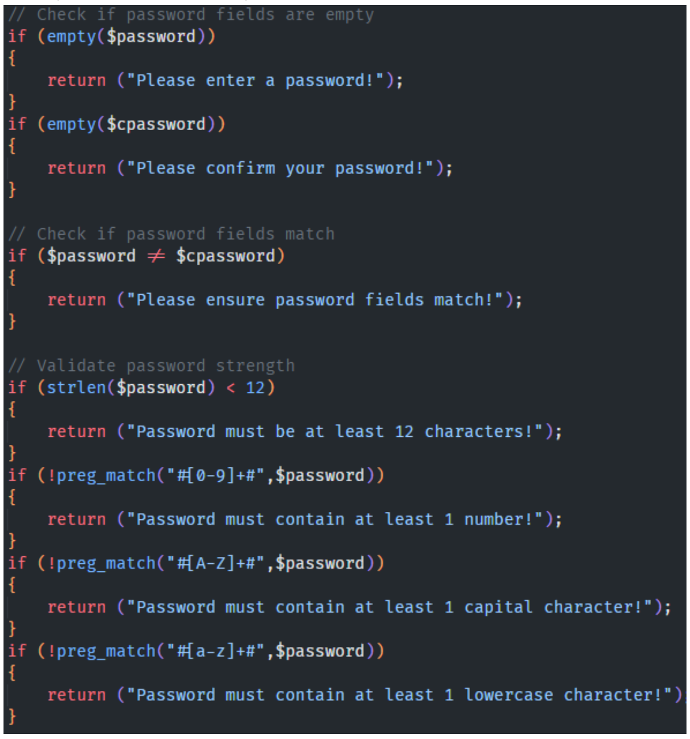
Why do you think it is secure? Use bullet points to provide your reasons and back it up with code snippet from your appliction.

- The arguments given by the user are sanitised:



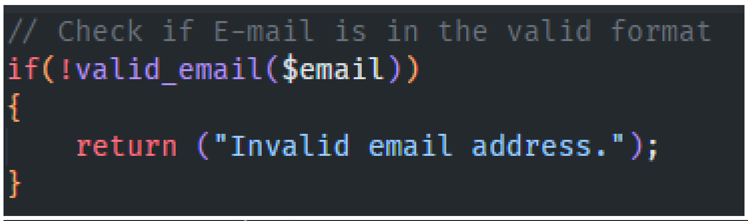
```
$args = func_get_args();

// Remove whitespace from arguments
$args = array_map(function($value){
    return trim($value);
}, $args);

// Prevent empty fields
foreach ($args as $value) {
    if(empty($value)){
        return ("all fields are required");
    }
}

// Prevent <> characters from input fields
foreach ($args as $value) {
    if (preg_match("/([ <|> ])/", $value)) {
        return ("<> characters are invalid");
    }
}
```

- The password must pass validation:

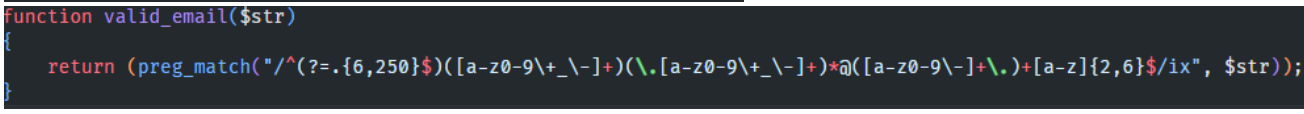


```
// Check if password fields are empty
if (empty($password))
{
    return ("Please enter a password!");
}
if (empty($cpassword))
{
    return ("Please confirm your password!");
}

// Check if password fields match
if ($password != $cpassword)
{
    return ("Please ensure password fields match!");
}

// Validate password strength
if (strlen($password) < 12)
{
    return ("Password must be at least 12 characters!");
}
if (!preg_match("#[0-9]+#",$password))
{
    return ("Password must contain at least 1 number!");
}
if (!preg_match("#[A-Z]+#",$password))
{
    return ("Password must contain at least 1 capital character!");
}
if (!preg_match("#[a-z]+#",$password))
{
    return ("Password must contain at least 1 lowercase character!")
}
```

- Email addresses must be in the correct format :

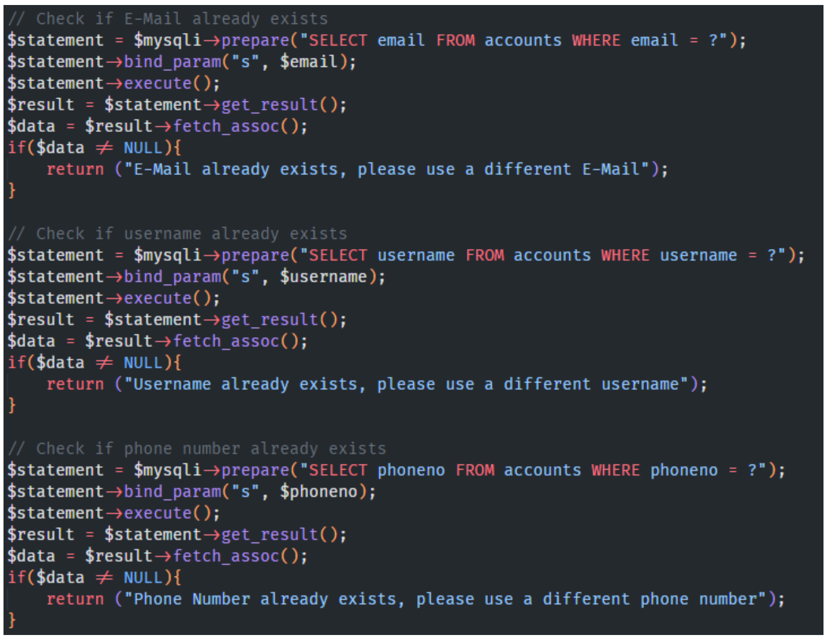


```
// Check if E-mail is in the valid format
if(!valid_email($email))
{
    return ("Invalid email address.");
}
```



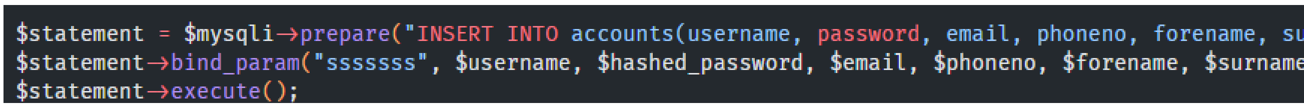
```
function valid_email($str)
{
    return (preg_match("/^(?=.{6,250}$)([a-z0-9\+_\-]+)(\.[a-z0-9\+_\-]+)*@([a-z0-9\-]+\.)+[a-z]{2,6}$/ix", $str));
}
```

- Only unique email, username and phone numbers may be used:



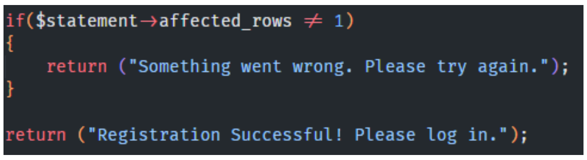
```
// Check if E-Mail already exists
$statement = $mysqli→prepare("SELECT email FROM accounts WHERE email = ?");
$statement→bind_param("s", $email);
$statement→execute();
$result = $statement→get_result();
$data = $result→fetch_assoc();
if($data ≠ NULL){
    return ("E-Mail already exists, please use a different E-Mail");
}

// Check if username already exists
$statement = $mysqli→prepare("SELECT username FROM accounts WHERE username = ?");
$statement→bind_param("s", $username);
$statement→execute();
$result = $statement→get_result();
$data = $result→fetch_assoc();
if($data ≠ NULL){
    return ("Username already exists, please use a different username");
}

// Check if phone number already exists
$statement = $mysqli→prepare("SELECT phoneno FROM accounts WHERE phoneno = ?");
$statement→bind_param("s", $phoneno);
$statement→execute();
$result = $statement→get_result();
$data = $result→fetch_assoc();
if($data ≠ NULL){
    return ("Phone Number already exists, please use a different phone number");
}
```

- The SQL is then prepared and the values once passing sanitation and validation get inserted into the database:



```
$statement = $mysqli→prepare("INSERT INTO accounts(username, password, email, phoneno, forename, su
$statement→bind_param("sssssss", $username, $hashed_password, $email, $phoneno, $forename, $surname
$statement→execute();
```

- If no rows are changed, then the user is asked to try again:

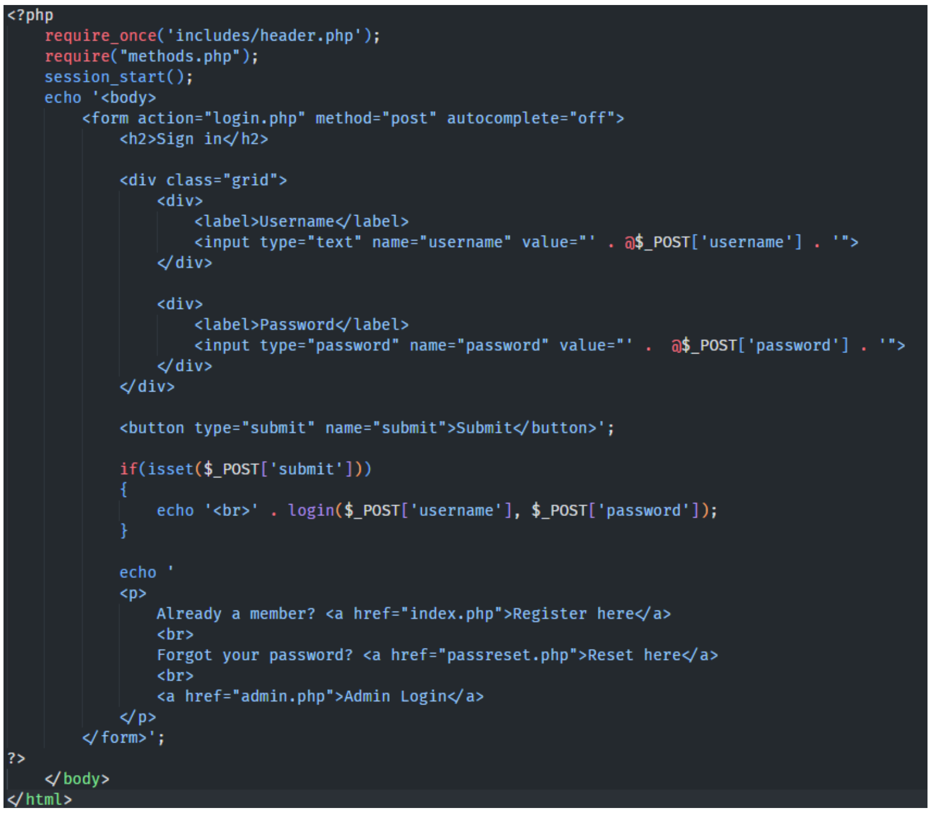


```
if($statement→affected_rows ≠ 1)
{
    return ("Something went wrong. Please try again.");
}

return ("Registration Successful! Please log in.");
```

Candidate No: 215709

Task 2 - Develop a secure login feature.

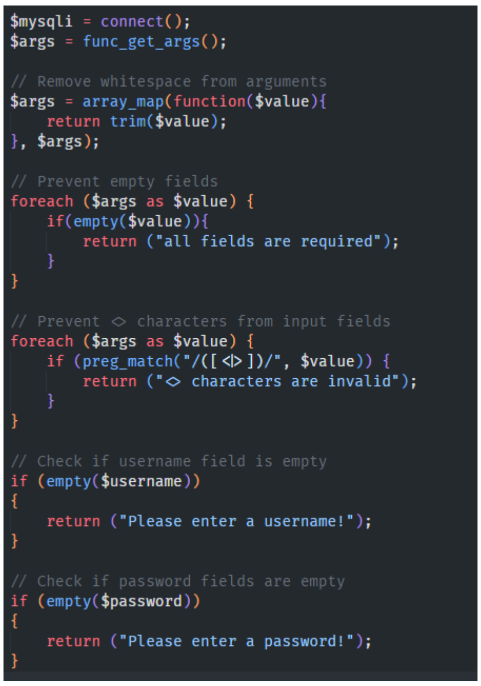
Login feature code screenshots



```
<?php
    require_once('includes/header.php');
    require("methods.php");
    session_start();
    echo '<body>
        <form action="login.php" method="post" autocomplete="off">
            <h2>Sign in</h2>

            <div class="grid">
                <div>
                    <label>Username</label>
                    <input type="text" name="username" value="' . @$_POST['username'] . '">
                </div>

                <div>
                    <label>Password</label>
                    <input type="password" name="password" value="' .  @$_POST['password'] . '">
                </div>
            </div>

            <button type="submit" name="submit">Submit</button>';

            if(isset($_POST['submit']))
            {
                echo '<br>' . login($_POST['username'], $_POST['password']);
            }

            echo '
            <p>
                Already a member? <a href="index.php">Register here</a>
                <br>
                Forgot your password? <a href="passreset.php">Reset here</a>
                <br>
                <a href="admin.php">Admin Login</a>
            </p>
        </form>';
?>
    </body>
</html>
```

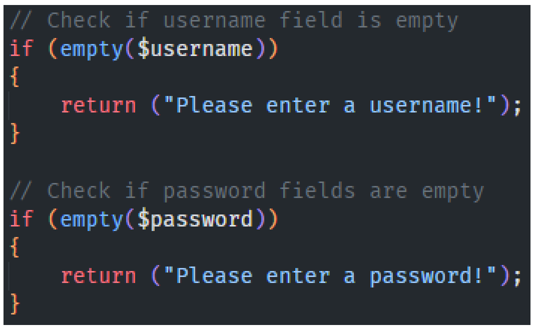
Why do you think it is secure? Use bullet points to provide your reasons and back it up code snippet from your application.

- User input is sanitised:



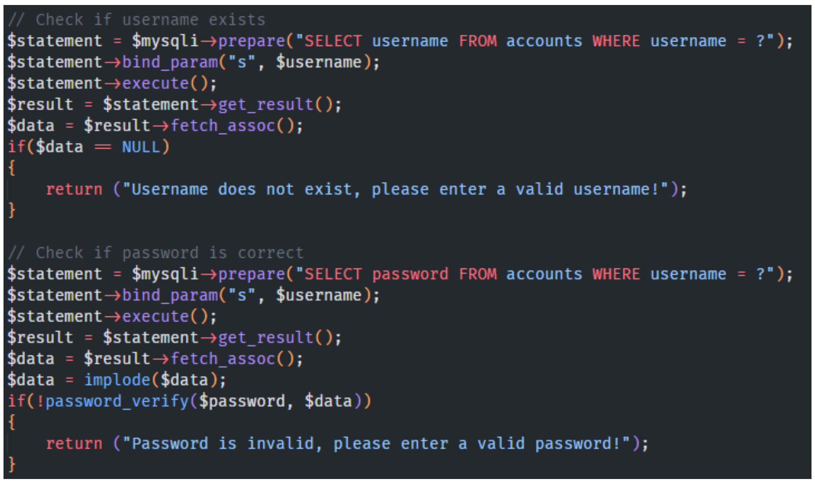
```
$mysqli = connect();
$args = func_get_args();

// Remove whitespace from arguments
$args = array_map(function($value){
    return trim($value);
}, $args);

// Prevent empty fields
foreach ($args as $value) {
    if(empty($value)){
        return ("all fields are required");
    }
}

// Prevent <> characters from input fields
foreach ($args as $value) {
    if (preg_match("/([<>])/", $value)) {
        return ("<> characters are invalid");
    }
}

// Check if username field is empty
if (empty($username))
{
    return ("Please enter a username!");
}

// Check if password fields are empty
if (empty($password))
{
    return ("Please enter a password!");
}
```

- Empty fields are not allowed:



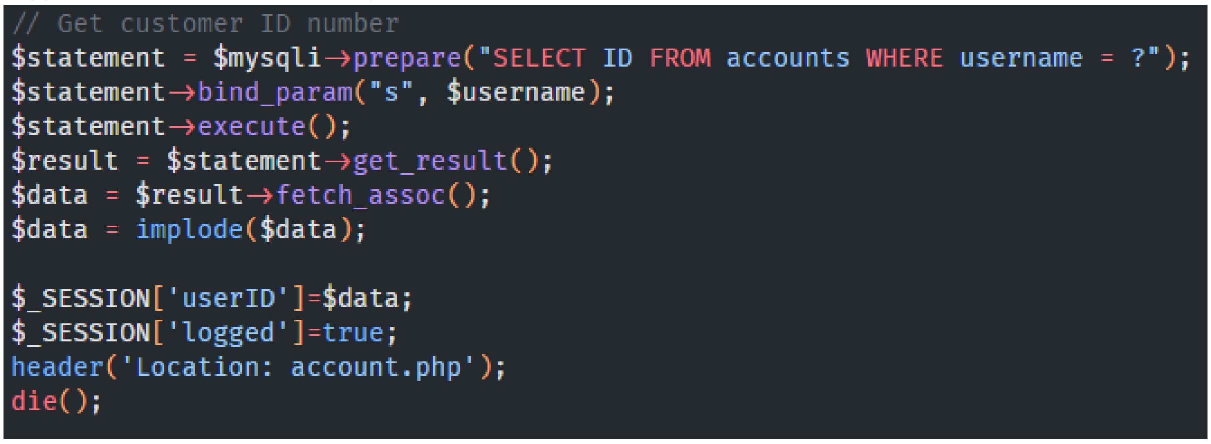
```php
// Check if username field is empty
if (empty($username))
{
    return ("Please enter a username!");
}

// Check if password fields are empty
if (empty($password))
{
    return ("Please enter a password!");
}
```

- Check if username and password match:



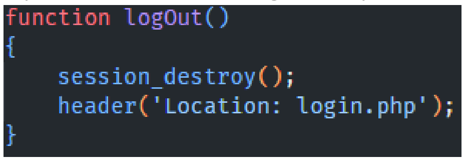
```php
// Check if username exists
$statement = $mysqli->prepare("SELECT username FROM accounts WHERE username = ?");
$statement->bind_param("s", $username);
$statement->execute();
$result = $statement->get_result();
$data = $result->fetch_assoc();
if($data == NULL)
{
    return ("Username does not exist, please enter a valid username!");
}

// Check if password is correct
$statement = $mysqli->prepare("SELECT password FROM accounts WHERE username = ?");
$statement->bind_param("s", $username);
$statement->execute();
$result = $statement->get_result();
$data = $result->fetch_assoc();
$data = implode($data);
if(!password_verify($password, $data))
{
    return ("Password is invalid, please enter a valid password!");
}
```

- User ID is stored as a session variable and the variable “logged” keeps track that the user has logged into the site successfully:



```php
// Get customer ID number
$statement = $mysqli->prepare("SELECT ID FROM accounts WHERE username = ?");
$statement->bind_param("s", $username);
$statement->execute();
$result = $statement->get_result();
$data = $result->fetch_assoc();
$data = implode($data);

$_SESSION['userID']=$data;
$_SESSION['logged']=true;
header('Location: account.php');
die();
```

- Option for a user to log out is permitted via account management page:



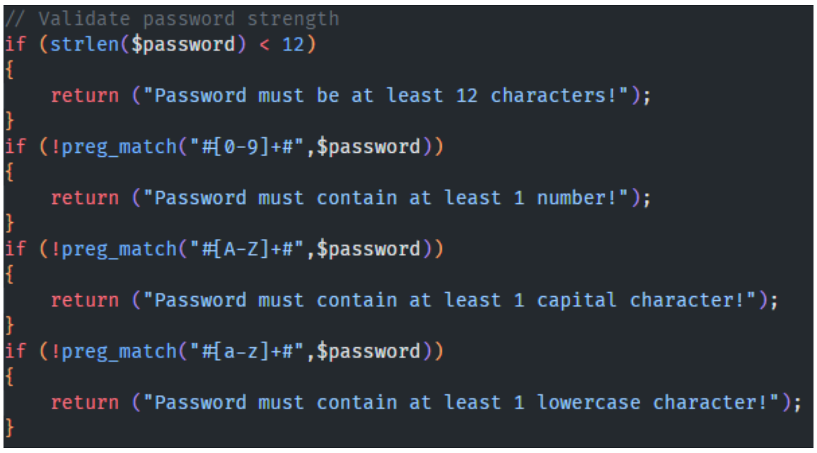
```php
function logOut()
{
    session_destroy();
    header('Location: login.php');
}
```

Task 3 - Implement password strength and password recovery

List each password policy element that you implemented and back it up with code snippets from your application

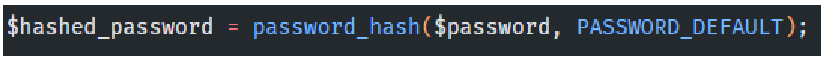
Guard clauses prevent invalid passwords that do not fit the requirements:

- 12 characters minimum
- At least one uppercase character
- At least one lowercase character
- At least one number



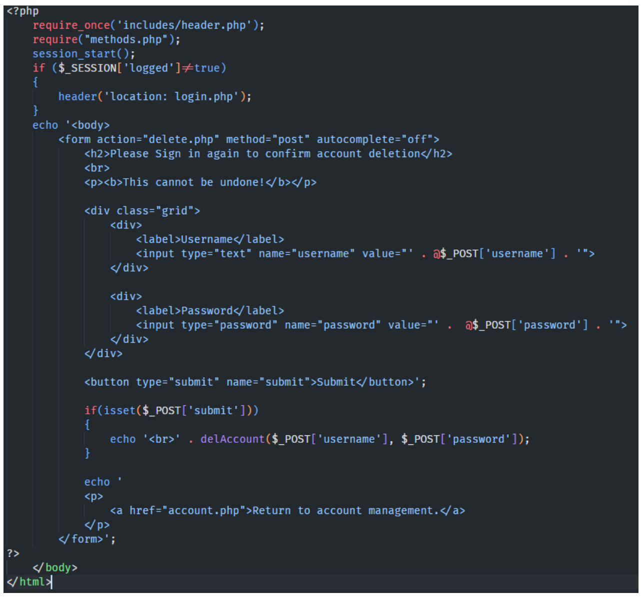
```
// Validate password strength
if (strlen($password) < 12)
{
    return ("Password must be at least 12 characters!");
}
if (!preg_match("#[0-9]+#",$password))
{
    return ("Password must contain at least 1 number!");
}
if (!preg_match("#[A-Z]+#",$password))
{
    return ("Password must contain at least 1 capital character!");
}
if (!preg_match("#[a-z]+#",$password))
{
    return ("Password must contain at least 1 lowercase character!");
}
```

Passwords are hashed with a salt to prevent dictionary or rainbow table attacks



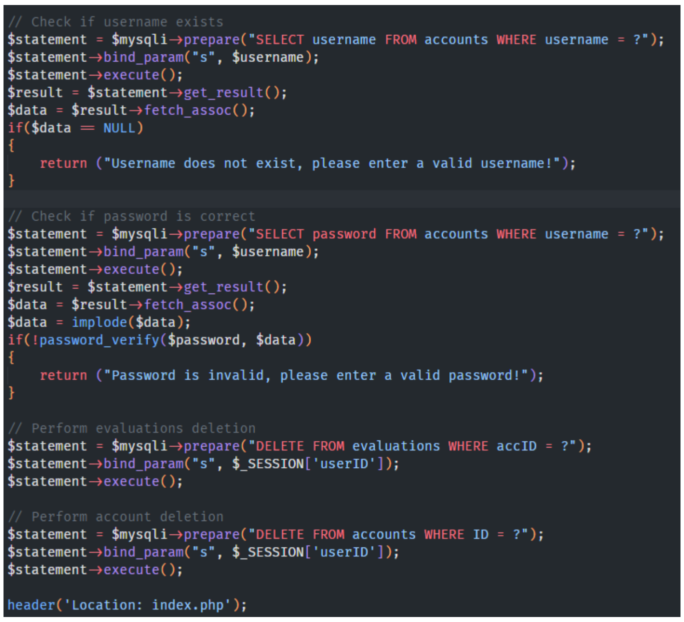
```
$hashed_password = password_hash($password, PASSWORD_DEFAULT);
```

At any time, a user can delete there account and it will remove their details from both the accounts table and all evaluations they have submitted



```
<?php
    require_once('includes/header.php');
    require("methods.php");
    session_start();
    if ($_SESSION['logged']≠true)
    {
        header('location: login.php');
    }
    echo '<body>
        <form action="delete.php" method="post" autocomplete="off">
            <h2>Please Sign in again to confirm account deletion</h2>
            <br>
            <p><b>This cannot be undone!</b></p>

            <div class="grid">
                <div>
                    <label>Username</label>
                    <input type="text" name="username" value="' . @$_POST['username'] . '">
                </div>

                <div>
                    <label>Password</label>
                    <input type="password" name="password" value="' .  @$_POST['password'] . '">
                </div>
            </div>

            <button type="submit" name="submit">Submit</button>';

            if(isset($_POST['submit']))
            {
                echo '<br>' . delAccount($_POST['username'], $_POST['password']);
            }

            echo '
            <p>
                <a href="account.php">Return to account management.</a>
            </p>
        </form>';
?>
    </body>
</html>
```

After sanitising the input fields again, the user must log in and they can then remove their data through a prepared statement

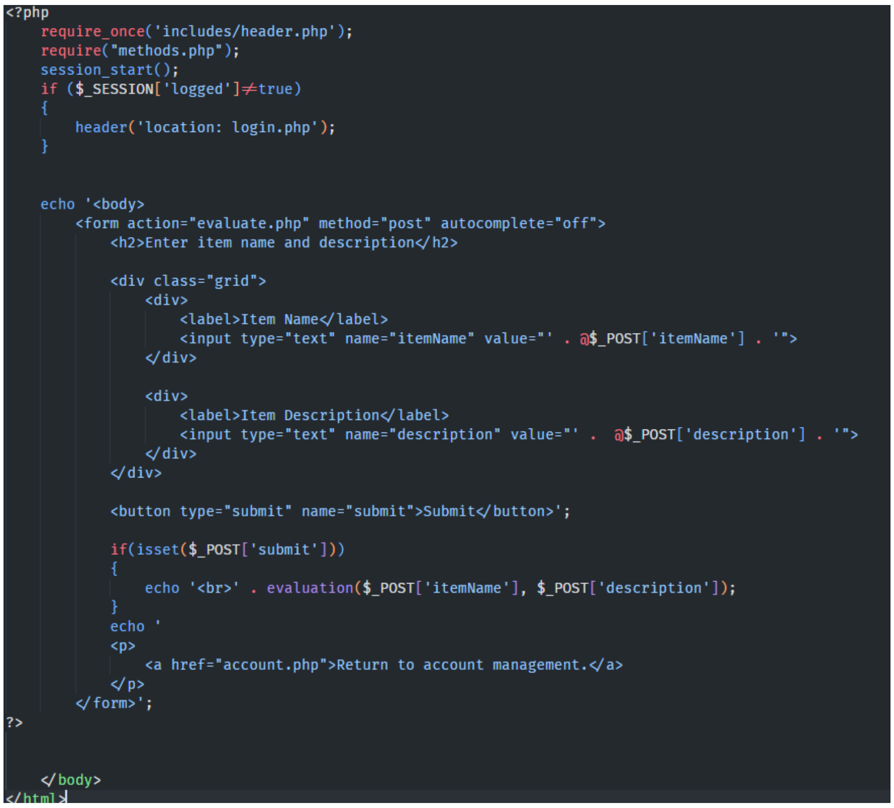


```php
// Check if username exists
$statement = $mysqli->prepare("SELECT username FROM accounts WHERE username = ?");
$statement->bind_param("s", $username);
$statement->execute();
$result = $statement->get_result();
$data = $result->fetch_assoc();
if($data == NULL)
{
    return ("Username does not exist, please enter a valid username!");
}

// Check if password is correct
$statement = $mysqli->prepare("SELECT password FROM accounts WHERE username = ?");
$statement->bind_param("s", $username);
$statement->execute();
$result = $statement->get_result();
$data = $result->fetch_assoc();
$data = implode($data);
if(!password_verify($password, $data))
{
    return ("Password is invalid, please enter a valid password!");
}

// Perform evaluations deletion
$statement = $mysqli->prepare("DELETE FROM evaluations WHERE accID = ?");
$statement->bind_param("s", $_SESSION['userID']);
$statement->execute();

// Perform account deletion
$statement = $mysqli->prepare("DELETE FROM accounts WHERE ID = ?");
$statement->bind_param("s", $_SESSION['userID']);
$statement->execute();

header('Location: index.php');
```

Candidate No: 215709

Task 4 - Implement a "Evaluation Request" web page.

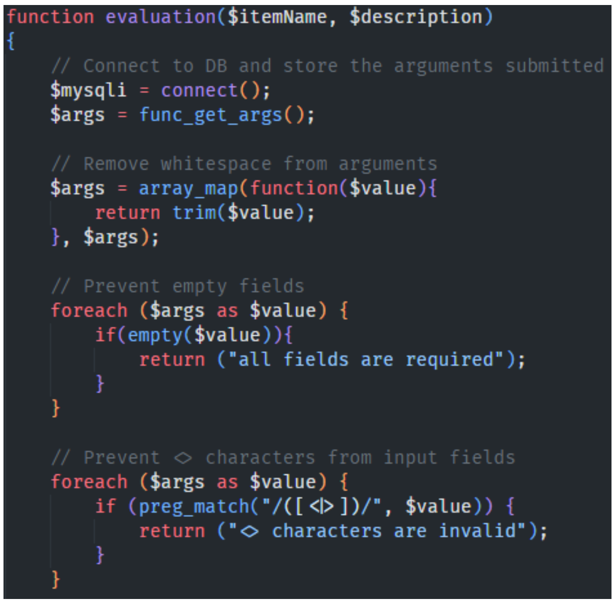
Request Evaluation feature screenshot



```php
<?php
    require_once('includes/header.php');
    require("methods.php");
    session_start();
    if ($_SESSION['logged']≠true)
    {
        header('location: login.php');
    }


    echo '<body>
        <form action="evaluate.php" method="post" autocomplete="off">
            <h2>Enter item name and description</h2>

            <div class="grid">
                <div>
                    <label>Item Name</label>
                    <input type="text" name="itemName" value="' . @$_POST['itemName'] . '">
                </div>

                <div>
                    <label>Item Description</label>
                    <input type="text" name="description" value="' .  @$_POST['description'] . '">
                </div>
            </div>

            <button type="submit" name="submit">Submit</button>';

            if(isset($_POST['submit']))
            {
                echo '<br>' . evaluation($_POST['itemName'], $_POST['description']);
            }
            echo '
            <p>
                <a href="account.php">Return to account management.</a>
            </p>
        </form>';
?>


    </body>
</html>
```

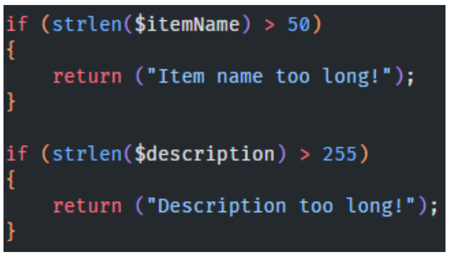
Why do you think it is secure?

As stated in the previous tasks, the input fields are sanitised



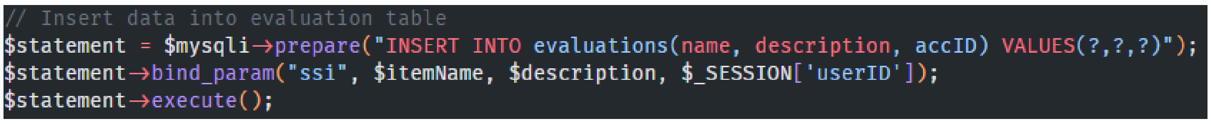
```php
function evaluation($itemName, $description)
{
    // Connect to DB and store the arguments submitted
    $mysqli = connect();
    $args = func_get_args();

    // Remove whitespace from arguments
    $args = array_map(function($value){
        return trim($value);
    }, $args);

    // Prevent empty fields
    foreach ($args as $value) {
        if(empty($value)){
            return ("all fields are required");
        }
    }

    // Prevent <> characters from input fields
    foreach ($args as $value) {
        if (preg_match("/([<>])/", $value)) {
            return ("<> characters are invalid");
        }
    }
```

Then the input size of the item name and description are capped



```php
if (strlen($itemName) > 50)
{
    return ("Item name too long!");
}

if (strlen($description) > 255)
{
    return ("Description too long!");
}
```

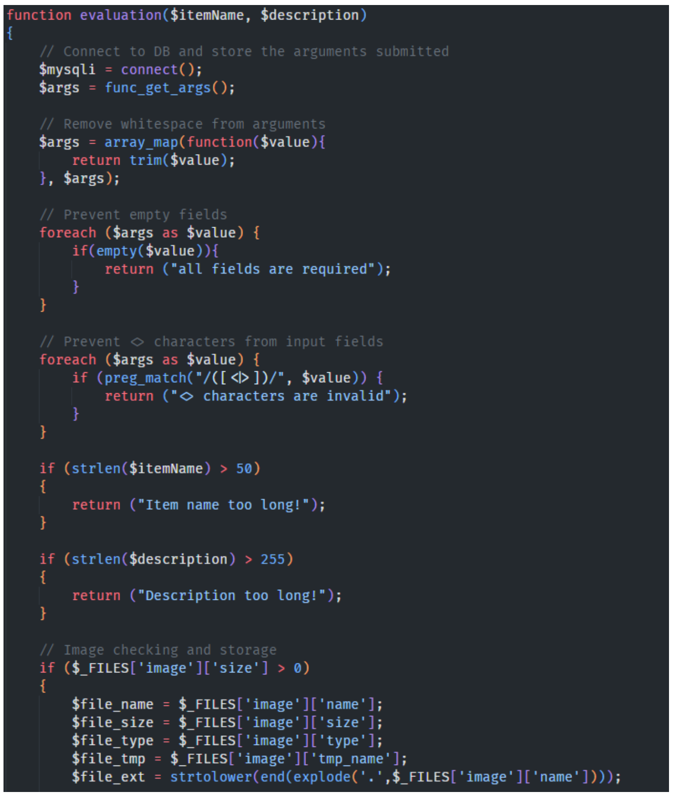
Finally, the data is prepared in an SQL statement and sent to the database



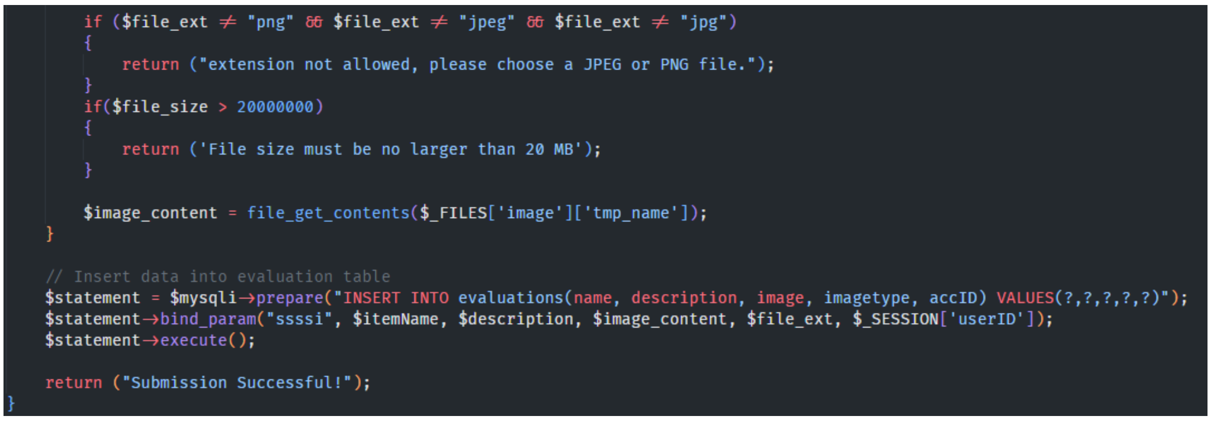
```php
// Insert data into evaluation table
$statement = $mysqli->prepare("INSERT INTO evaluations(name, description, accID) VALUES(?,?,?)");
$statement->bind_param("ssi", $itemName, $description, $_SESSION['userID']);
$statement->execute();
```

Task 5 - Develop a feature that will allow customers to submit photographs

Code of the feature



```php
function evaluation($itemName, $description)
{
    // Connect to DB and store the arguments submitted
    $mysqli = connect();
    $args = func_get_args();

    // Remove whitespace from arguments
    $args = array_map(function($value){
        return trim($value);
    }, $args);

    // Prevent empty fields
    foreach ($args as $value) {
        if(empty($value)){
            return ("all fields are required");
        }
    }

    // Prevent <> characters from input fields
    foreach ($args as $value) {
        if (preg_match("/([<|>])/", $value)) {
            return ("<> characters are invalid");
        }
    }

    if (strlen($itemName) > 50)
    {
        return ("Item name too long!");
    }

    if (strlen($description) > 255)
    {
        return ("Description too long!");
    }

    // Image checking and storage
    if ($_FILES['image']['size'] > 0)
    {
        $file_name = $_FILES['image']['name'];
        $file_size = $_FILES['image']['size'];
        $file_type = $_FILES['image']['type'];
        $file_tmp = $_FILES['image']['tmp_name'];
        $file_ext = strtolower(end(explode('.',$_FILES['image']['name'])));
```



```php
        if ($file_ext != "png" && $file_ext != "jpeg" && $file_ext != "jpg")
        {
            return ("extension not allowed, please choose a JPEG or PNG file.");
        }
        if($file_size > 20000000)
        {
            return ('File size must be no larger than 20 MB');
        }

        $image_content = file_get_contents($_FILES['image']['tmp_name']);
    }

    // Insert data into evaluation table
    $statement = $mysqli->prepare("INSERT INTO evaluations(name, description, image, imagetype, accID) VALUES(?,?,?,?,?)");
    $statement->bind_param("ssssi", $itemName, $description, $image_content, $file_ext, $_SESSION['userID']);
    $statement->execute();

    return ("Submission Successful!");
}
```

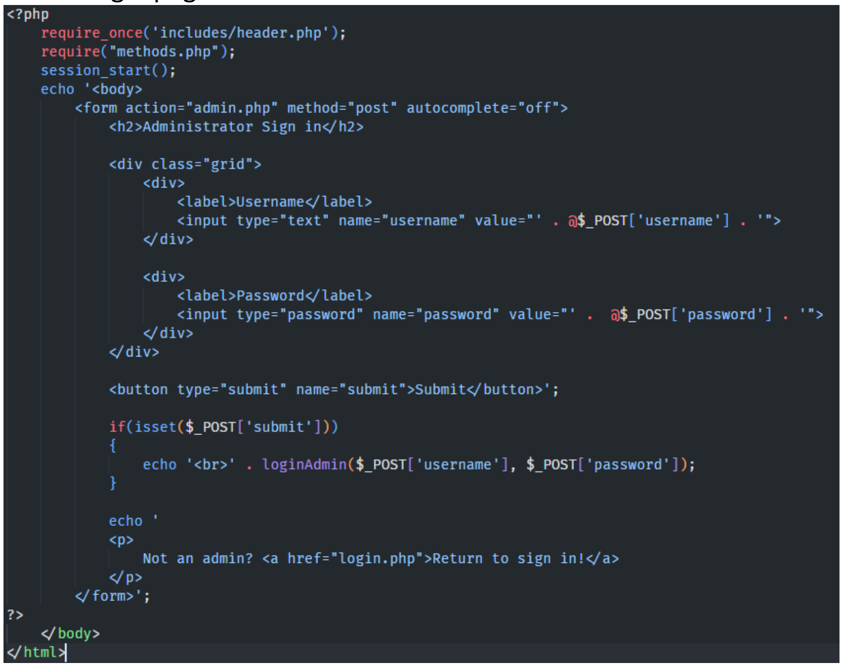
Why do I think it is secure?

- File extension is checked to prevent malicious files
- File size is limited to only 20MB
- Input is sanitised for the text portion
- No files are run
- Images are stored as a string inside of a longblob
- SQL statement is prepared before insertion to prevent SQL injection

Task 6 – Request Listing Page

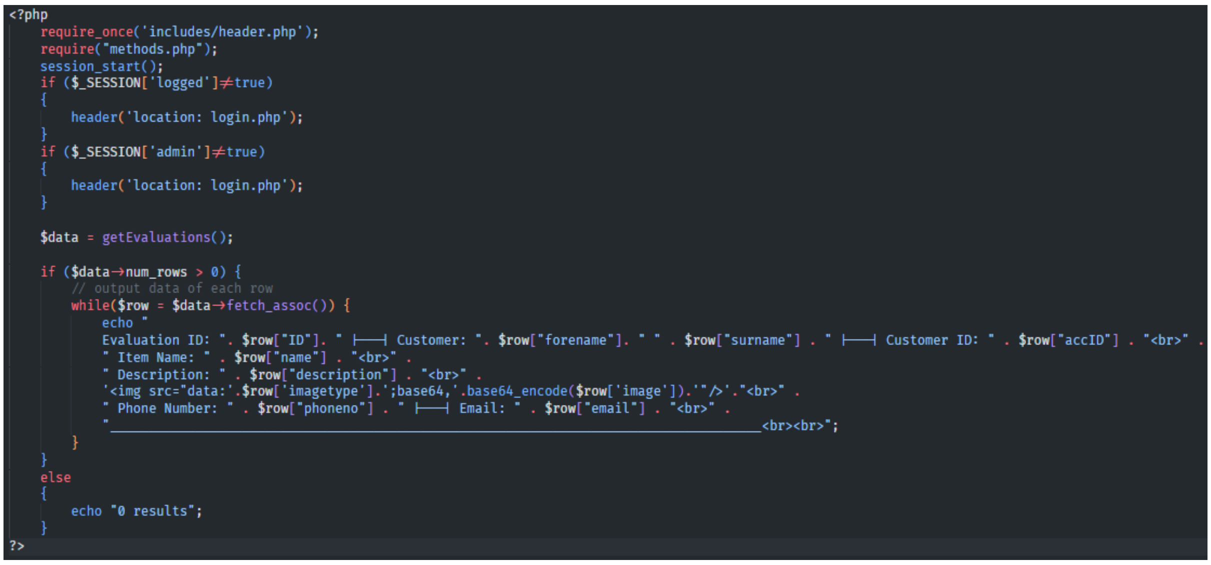
Code of the feature

Admin login page



```php
<?php
    require_once('includes/header.php');
    require("methods.php");
    session_start();
    echo '<body>
        <form action="admin.php" method="post" autocomplete="off">
            <h2>Administrator Sign in</h2>

            <div class="grid">
                <div>
                    <label>Username</label>
                    <input type="text" name="username" value="' . @$_POST['username'] . '">
                </div>

                <div>
                    <label>Password</label>
                    <input type="password" name="password" value="' .  @$_POST['password'] . '">
                </div>
            </div>

            <button type="submit" name="submit">Submit</button>';

            if(isset($_POST['submit']))
            {
                echo '<br>' . loginAdmin($_POST['username'], $_POST['password']);
            }

            echo '
            <p>
                Not an admin? <a href="login.php">Return to sign in!</a>
            </p>
        </form>';
?>
    </body>
</html>
```

Evaluation display page



```php
<?php
    require_once('includes/header.php');
    require("methods.php");
    session_start();
    if ($_SESSION['logged']≠true)
    {
        header('location: login.php');
    }
    if ($_SESSION['admin']≠true)
    {
        header('location: login.php');
    }

    $data = getEvaluations();

    if ($data→num_rows > 0) {
        // output data of each row
        while($row = $data→fetch_assoc()) {
            echo "
            Evaluation ID: ". $row["ID"]. " ├──┤ Customer: ". $row["forename"]. " " . $row["surname"] . " ├──┤ Customer ID: " . $row["accID"] . "<br>" .
            " Item Name: " . $row["name"] . "<br>" .
            " Description: " . $row["description"] . "<br>" .
            '<img src="data:'.$row['imagetype'].';base64,'.base64_encode($row['image']).'"/>'."<br>" .
            " Phone Number: " . $row["phoneno"] . " ├──┤ Email: " . $row["email"] . "<br>" .
            "______________________________________________________________________________<br><br>";
        }
    }
    else
    {
        echo "0 results";
    }
?>
```

Why is it secure?

- Session allows only authorised users onto the page
- Only a user with a true value in the admin Boolean variable can log in here
- Inputs are sanitised
- Account passwords of the users are kept hidden from any administrator