

INTI International College Penang

School of Engineering and Technology

3+0 Bachelor of Science (Hons) in Computer Science, in collaboration with Coventry University, UK
3+0 Bachelor of Science (Hons) in Computing, in collaboration with Coventry University, UK


Coursework cover sheet

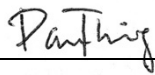
Section A - To be completed by the student

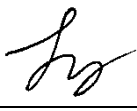
Full Name:	
Group Member 1: Damien Tan Lek Khee	
Group Member 2: Lim Pau Thing	
Group Member 3: Ooi Ying Jie	
CU Student ID Number:	
Group Member 1: 12672844	
Group Member 2: 12672877	
Group Member 3: 12672888	
Semester	: 10
Lecturer	: R.K.Krishnamoorthy
Module Code and Title	: INT6005CEM Security
Assignment No. / Title	: Secure Application Development
	50% of Module Mark
Hand out date: 5/10/23	Due date : 23/11/23
Penalties: No late work will be accepted. If you are unable to submit coursework on time due to extenuating circumstances, you may be eligible for an extension. Please consult the lecturer.	

Declaration: I/we the undersigned confirm that I/we have read and agree to abide by the University regulations on plagiarism and cheating and Faculty coursework policies and procedures. I/we confirm that this piece of work is my/our own. I/we consent to appropriate storage of our work for plagiarism checking.

Signature(s):

Group Member 1: 

Group Member 2: 

Group Member 3: 

Section B - To be completed by the module leader

Intended learning outcomes assessed by this work:		
1. Develop and evaluate software that addresses the most common and most severe security concerns.		
2. Critically evaluate the security of an IT ecosystem.		
Marking scheme	Max	Mark
1. Report <i>Refer to the section on Marking Scheme on detailed breakdown.</i>	60	
2. System Functionality <i>Refer to the section on Marking Scheme on detailed breakdown.</i>	40	
Total	100	
<u>Lecturer's Feedback</u>		
<u>Internal Moderator's Feedback</u>		

Table of Contents

Table of Contents.....	4
1.0 Introduction and Overview	7
2.0 Design	8
2.1 Potential security issues (for each element of design).....	8
2.2 Recommendations (for each potential security issues).....	9
3.0 Implementation	10
3.1 Security Considerations and Implementation	10
3.2 Security Techniques and Methods	11
4.0 Discussion	31
4.1 Key Findings and Issue Resolution.....	31
4.2 Continuous Improvement and Maintenance of Security	32
5.0 Summary	34
5.1 Conclusion	34
5.2 Learning Outcome Achieved	34
6.0 References.....	35
7.0 Appendix.....	36

Table of Figures

Figure 1: Food Ordering and Management System	7
Figure 3: Input Validation (Login Check)	11
Figure 4: Input Validation (Category Add).....	11
Figure 5: Input Validation (Category Edit).....	11
Figure 6: Input Validation (Food Edit)	12
Figure 7: Input Validation (Staff Edit).....	12
Figure 8: Input Validation (Table Edit)	12
Figure 9: Input Validation (Profile Edit).....	13
Figure 10: Input Validation (Waitlist Add).....	13
Figure 11: Input Validation (Waitlist Edit).....	14
Figure 12: Password Verification & Hashing	14
Figure 13: SQL Statement Modifications (Category Update)	15
Figure 14: SQL Statement Modifications (Category Delete)	16
Figure 15: SQL Statement Modifications (Food Insert)	17
Figure 16: SQL Statement Modifications (Food Update).....	17
Figure 17: SQL Statement Modifications (Food Delete).....	18
Figure 18: SQL Statement Modifications (Login Check).....	18
Figure 19: SQL Statement Modifications (Staff Delete)	19
Figure 20: SQL Statement Modifications (Staff Insert).....	19
Figure 21: SQL Statement Modifications (Staff Update)	20
Figure 22: SQL Statement Modifications (Order Delete).....	20
Figure 23: SQL Statement Modifications (Order Update).....	21
Figure 24: SQL Statement Modifications (Order Insert)	21
Figure 25: SQL Statement Modifications (Table Delete).....	22
Figure 26: SQL Statement Modifications (Table Update).....	22
Figure 27: SQL Statement Modifications (Table Insert)	23
Figure 28: SQL Statement Modifications (Tax Delete).....	23
Figure 29: SQL Statement Modifications (Tax Insert)	24
Figure 30: SQL Statement Modifications (Tax Update).....	24
Figure 31: SQL Statement Modifications (Wait Insert).....	25
Figure 32: SQL Statement Modifications (Wait Update)	25
Figure 33: SQL Statement Modifications (Wait Delete)	26

Figure 34: SQL Statement Modifications (Profile Update)	27
Figure 35: reCAPTCHA Integration	28
Figure 36: reCAPTCHA Container	29
Figure 37: reCAPTCHA Integration (Login Check)	30

1.0 Introduction and Overview

Due to its handling of sensitive data and user interactions, the food ordering and reservation management system created for restaurant staff places a high priority on security. The security measures put in place during the system's design and development to guarantee data integrity, availability, and privacy are described in this report. Robust security measures, such as error handling, input validation, vulnerability protection, and CAPTCHA integration, have been implemented to protect sensitive data and guarantee the application's correct operation when staff members use it for reservations and meal orders.

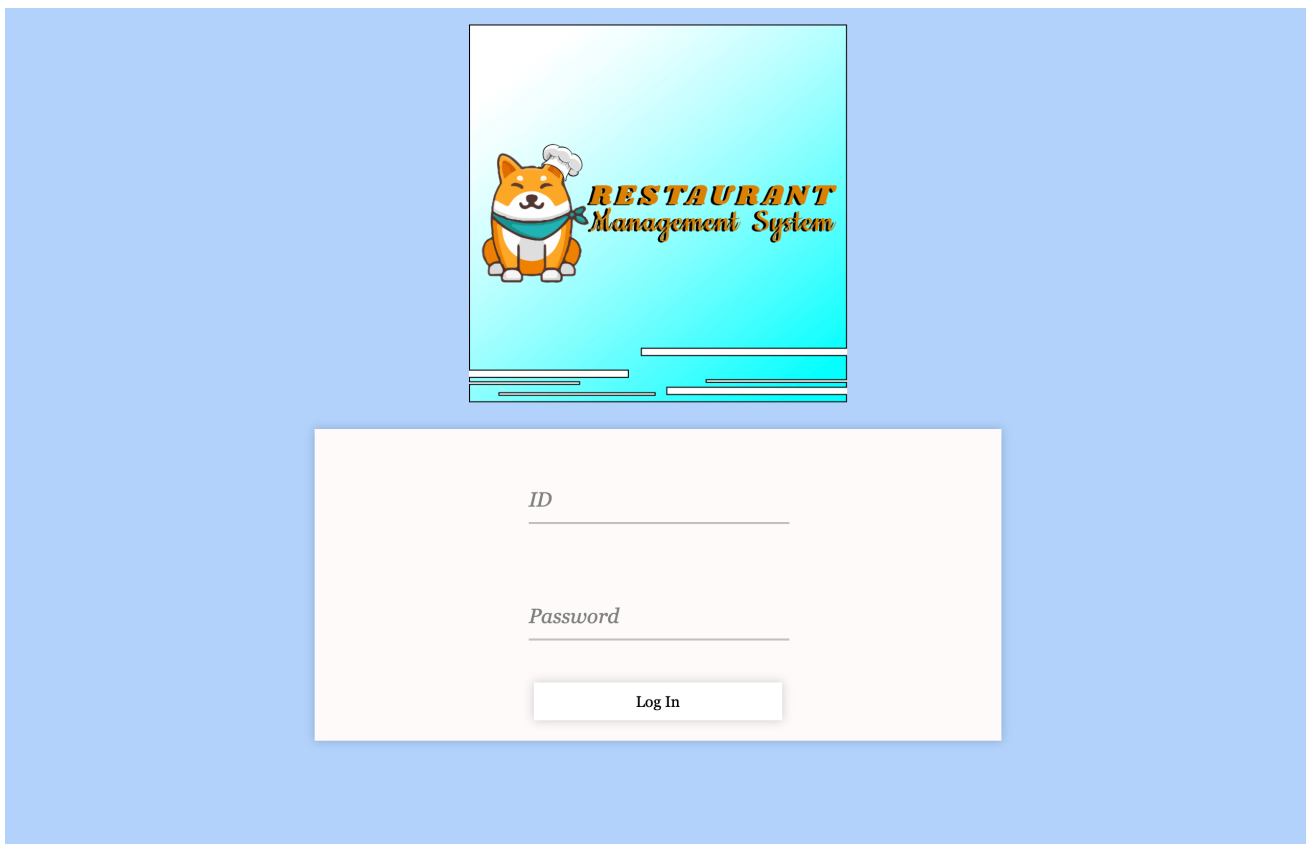


Figure 1: Food Ordering and Management System

2.0 Design

2.1 Potential security issues (for each element of design)

a. Inadequate Input Validation and Error Handling

Inadequate input validation raises a security risk because it can result in SQL injection or other types of attacks.

b. Passwords Stored in PlainText

Without adequate password hashing mechanisms, there is a chance that sensitive data could be accessed by unauthorized parties.

c. SQL Injection Vulnerabilities in Statements

Malicious actors may alter database queries in the absence of parameterized queries or prepared statements, which could result in unauthorized data access.

d. Absence of CAPTCHA

The absence of CAPTCHA could leave the system vulnerable to automated attacks.

2.2 Recommendations (for each potential security issues)

a. Inadequate Input Validation and Error Handling

Implementation of thorough input validation codes to sanitize user inputs and prevent SQL injection or other malicious attacks. Additionally, error handling codes will be modified and added.

b. Passwords Stored in PlainText

Integration of reliable algorithm to enhance existing password hashing codes.

c. SQL Injection Vulnerabilities in Statements

Modifications of existing SQL statements by adding parameterized queries or prepared statements. Constant updates of database access credentials and restrict access to authorized personnel only.

d. Absence of CAPTCHA

Implementation of CAPTCHA codes hence preventing automated attacks during user login.

3.0 Implementation

3.1 Security Considerations and Implementation

a. Comprehensive Input Validation and Error Handling

The implementation addresses security risks by incorporating robust input validation to prevent malicious attacks like SQL injection. Error handling codes are also enhanced to ensure clear messages without compromising confidential data.

b. Integration of Reliable Password Hashing Algorithm

The implementation concentrates on the integration of a trustworthy algorithm to improve the current password hashing codes to counter the risk associated with passwords stored in plaintext. This guarantees the security of sensitive data by lowering the possibility of unauthorized access using a reliable and tested hashing algorithm.

c. Modifications of SQL Statements

Security is improved by modifying SQL statements to mitigate vulnerability to SQL injection. Prepared statements and restricted access for authorized personnel contribute significantly to lowering the risk of unauthorized data access. Regular updating of database access credentials further enhances overall system security.

d. Integration of CAPTCHA Codes

The implementation plan calls for the integration of CAPTCHA codes to reduce the risk associated with the lack of CAPTCHA. By adding this, automated attacks against the user registration and login processes are thwarted.

3.2 Security Techniques and Methods

a. Comprehensive Input Validation and Error Handling (Ying Jie)

```
// Validate input fields to avoid empty values
if (empty($USER_id) || empty($USER_pwd)) {
    $_SESSION['message'] = "Please fill in all fields.";
    echo "<script>alert('Please fill in all fields.');
```

Figure 2: Input Validation (Login Check)

Ensures that both \$USER_id and \$USER_pwd must not be empty, hence reducing the risk of unauthorized access.

```
// Validate input fields to avoid empty values
if (empty($CATEGORY_name) || empty($CATEGORY_status)) {
    $_SESSION['message'] = "<script>alert('Please fill in all fields.');
```

Figure 3: Input Validation (Category Add)

Ensure that all \$CATEGORY_name and \$CATEGORY_status must not be empty. Special characters or numbers are considered invalid.

```
// Validate input fields to avoid empty values
if (empty($CATEGORY_status)) {
    $_SESSION['message'] = "<script>alert('Please select a status.');
```

Figure 4: Input Validation (Category Edit)

\$CATEGORY_status must be selected during the editing process to maintain data accuracy.

```
// Validate input fields to avoid empty values
if (empty($FOOD_status)) {
    $_SESSION['message'] = "<script>alert('Please select a status.');
```

Figure 5: Input Validation (Food Edit)

Ensure that \$FOOD_status must be selected.

```
// Validate input fields to avoid empty values
if (empty($POSITION)) {
    $_SESSION['message'] = "<script>alert('Please select a position.');
```

Figure 6: Input Validation (Staff Edit)

Ensures that the \$POSITION must be selected to maintain data accuracy.

```
// Validate input fields to avoid empty values
if (empty($STATUS)) {
    $_SESSION['message'] = "<script>alert('Please select a status.');
```

Figure 7: Input Validation (Table Edit)

Ensure that \$STATUS must be selected.

```
// Validate input fields to avoid empty values
if (empty($NAME) || empty($EMAIL) || empty($PASSWORD) || empty($CONTACT) || empty($BIRTHDAY) || empty($GENDER)) {
    $_SESSION['message'] = "<script>alert('Please fill in all fields.');

```

Figure 8: Input Validation (Profile Edit)

Ensures that user information such as name, email, password, contact information, birthday, and gender be chosen or entered. Data security is improved by specific restrictions on special characters and acceptable formats for phone numbers and emails.

```
// Validate phone number (assuming a simple format)
if (!preg_match('/^\d{10}$/', $CUS_contact)) {
    $_SESSION['message'] = "<script>alert('Invalid phone number.');

```

Figure 9: Input Validation (Waitlist Add)

Ensure that \$CUS_contact must be a valid phone number.

```
// Validate phone number (assuming a simple format)
if (!preg_match('/^\d{10}$/', $CONTACT)) {
    $_SESSION['message'] = "<script>alert('Invalid phone number.');

```

Figure 10: Input Validation (Waitlist Edit)

Similar to Waitlist add page, the \$CONTACT must also be a valid phone number during the editing process.

b. Integration of Reliable Password Hashing Algorithm (Ying Jie)

```
$HASHEDPWD = password_hash($PASSWORD, PASSWORD_DEFAULT);
```

```
// Verify the entered password against the hashed password in the database
if (password_verify($USER_pwd, $row['User_pwd'])) {
    $_SESSION['User_ID'] = $row['User_ID'];
    $DATETIME = $object->get_datetime();

    mysqli_query($connected, "INSERT INTO `attendance_table` (`Staff_ID`, `LoginTime`) VALUES ('".$_SESSION['User_ID']."', '$DATETIME');");
    header("location:Dashboard.php");
} else {
    $_SESSION['message'] = "Login failed. Please check your ID and password.";
    header("location:index.php?st=failure");
}
```

Figure 11: Password Verification & Hashing

This PHP code uses 'password_hash' with crypt to securely hash passwords. The hashed password is stored and compared during authentication. Successful verification logs login time and grants access to the dashboard. Failure results in a redirect with a status and message. The approach enhances security by ensuring proper user credentials for dashboard access.

c. Modifications of SQL Statements (Pau Thing)

```
$query = "UPDATE `category_table` SET  
        `Category_name` = ?,  
        `Category_status` = ?  
        WHERE `ID` = ?";  
$statement = mysqli_prepare($connected, $query);
```

```
// Check if the prepared statement is successfully updated  
if ($statement) {  
    // Check the number of affected rows after the update operation  
    mysqli_stmt_bind_param($statement, "ssi", $CATEGORY_name, $CATEGORY_status, $ID);  
  
    // Execute the statement  
    if (mysqli_stmt_execute($statement)) {  
        $rowsAffected = mysqli_stmt_affected_rows($statement);  
  
        if ($rowsAffected > 0) {  
            $_SESSION['message'] = "<script>alert('Category updated!');</script>";  
            header("location:Category.php?st=updated");  
        } else {  
            $_SESSION['message'] = "<script>alert('Category not found. Update failed. Please try again.');" .  
            header("location:Category.php?st=failure");  
        }  
    } else {  
        // echo "Error executing the statement: " . mysqli_stmt_error($statement);  
        $_SESSION['message'] = "<script>alert('Category update failed. Please Try again.');" .  
        header("location:Category.php?st=failure");  
    }  
}  
  
// Close the prepared statement  
mysqli_stmt_close($statement);
```

Figure 12: SQL Statement Modifications (Category Update)

```
$query = "DELETE FROM category_table WHERE ID = ?";  
$statement = mysqli_prepare($connected, $query);
```

```

// Check if the prepared statement is successfully deleted
if ($statement) {
    mysqli_stmt_bind_param($statement, "i", $DELETE_id);

    // Execute the statement
    if (mysqli_stmt_execute($statement)) {
        // Check the number of affected rows after the delete operation
        $rowsAffected = mysqli_stmt_affected_rows($statement);

        if ($rowsAffected > 0) {
            header("location:Category.php?st=Deleted");
        } else {
            $_SESSION['message'] = "<script>alert('Category delete failed. Please try again.');

```

Figure 13: SQL Statement Modifications (Category Delete)

```

$query_check = "SELECT * FROM menu_table WHERE Food_ID = ?";
$stmt_check = mysqli_prepare($connected, $query_check);

if ($stmt_check) {
    mysqli_stmt_bind_param($stmt_check, "s", $FOOD_id);
    mysqli_stmt_execute($stmt_check);

    mysqli_stmt_store_result($stmt_check);
    if (mysqli_stmt_num_rows($stmt_check) > 0) {
        $_SESSION['message'] = "<script>alert('Food ID already exists! Please use another Food ID.');

```

```

$query = "INSERT INTO `menu_table`
        (`Food_ID`, `Food_name`, `Food_cost`, `Category_name`, `Food_status`)
        VALUES (?, ?, ?, ?, ?)";
$stmt = mysqli_prepare($connected, $query);

```



```

if ($statement) {
    mysqli_stmt_bind_param($statement, "sssss", $FOOD_id, $FOOD_name, $FOOD_cost, $CATEGORY_name, $FOOD_status);

    if (mysqli_stmt_execute($statement)) {
        $_SESSION['message'] = "<script>alert('New food menu added !');</script>";
        header("location:FoodMenu.php?st=success");
    } else {
        $_SESSION['message'] = "<script>alert('Add new food menu failed. Please try again.');" . "</script>";
        header("location:FoodMenu.php?st=failure");
    }

    // Close the prepared statement
    mysqli_stmt_close($statement);
}

```

Figure 14: SQL Statement Modifications (Food Insert)

```

$query = "UPDATE `menu_table` SET
        `Food_name` = ?,
        `Food_cost` = ?,
        `Food_status` = ?
        WHERE `Food_ID` = ?";
$stmt = mysqli_prepare($connected, $query);

```

```

// Check if the prepared statement is successfully updated
if ($statement) {
    // Check the number of affected rows after the update operation
    mysqli_stmt_bind_param($statement, "sdsss", $FOOD_name, $FOOD_cost, $FOOD_status, $ID);

    // Execute the statement
    if (mysqli_stmt_execute($statement)) {
        $rowsAffected = mysqli_stmt_affected_rows($statement);

        if ($rowsAffected > 0) {
            $_SESSION['message'] = "<script>alert('Food menu updated !');</script>";
            header("location:FoodMenu.php?st=updated");
        } else {
            $_SESSION['message'] = "<script>alert('Food menu update failed. Please try again.');" . "</script>";
            header("location:FoodMenu.php?st=failure");
        }
    } else {
        // echo "Error executing the statement: " . mysqli_stmt_error($statement);
        $_SESSION['message'] = "<script>alert('Food menu update failed. Please Try again.');" . "</script>";
        header("location:FoodMenu.php?st=failure");
    }

    // Close the prepared statement
    mysqli_stmt_close($statement);
}

```

Figure 15: SQL Statement Modifications (Food Update)

```

// Check if the prepared statement is successfully updated
if ($statement) {
    // Check the number of affected rows after the update operation
    mysqli_stmt_bind_param($statement, "sdss", $FOOD_name, $FOOD_cost, $FOOD_status, $ID);

    // Execute the statement
    if (mysqli_stmt_execute($statement)) {
        $rowsAffected = mysqli_stmt_affected_rows($statement);

        if ($rowsAffected > 0) {
            $_SESSION['message'] = "<script>alert('Food menu updated !');</script>";
            header("location:FoodMenu.php?st=updated");
        } else {
            $_SESSION['message'] = "<script>alert('Food menu update failed. Please try again.');" . "</script>";
            header("location:FoodMenu.php?st=failure");
        }
    } else {
        // echo "Error executing the statement: " . mysqli_stmt_error($statement);
        $_SESSION['message'] = "<script>alert('Food menu update failed. Please Try again.');" . "</script>";
        header("location:FoodMenu.php?st=failure");
    }

    // Close the prepared statement
    mysqli_stmt_close($statement);
}

```

```

$query = "DELETE FROM menu_table WHERE Food_ID = ?";
$stmt = mysqli_prepare($connected, $query);

```

Figure 16: SQL Statement Modifications (Food Delete)

```

$query = "SELECT * FROM user_table WHERE User_ID = ?";
$stmt = mysqli_prepare($connected, $query);

if ($stmt) {
    mysqli_stmt_bind_param($stmt, "s", $USER_id);
    mysqli_stmt_execute($stmt);

    $result = mysqli_stmt_get_result($stmt);

    // Check if a row exists for the given User_ID
    if ($row = mysqli_fetch_assoc($result)) {

```

Figure 17: SQL Statement Modifications (Login Check)

```

$query = "DELETE FROM user_table WHERE User_ID = ?";
$stmt = mysqli_prepare($conn, $query);

// Check if the prepared statement is successfully deleted
if ($stmt) {
    mysqli_stmt_bind_param($stmt, "s", $DELETE_id);

    // Execute the statement
    if (mysqli_stmt_execute($stmt)) {
        // Check the number of affected rows after the delete operation
        $rowsAffected = mysqli_stmt_affected_rows($stmt);

        if ($rowsAffected > 0) {
            header("location:Staff.php?st=Deleted");
        } else {
            $_SESSION['message'] = "<script>alert('Staff remove failed. Please try again.');

```

Figure 18: SQL Statement Modifications (Staff Delete)

```

$query_check = "SELECT * FROM user_table WHERE User_ID = ?";
$stmt_check = mysqli_prepare($conn, $query_check);

if ($stmt_check) {
    mysqli_stmt_bind_param($stmt_check, "s", $FOOD_id);
    mysqli_stmt_execute($stmt_check);

    mysqli_stmt_store_result($stmt_check);
    if (mysqli_stmt_num_rows($stmt_check) > 0) {
        $_SESSION['message'] = "<script>alert('User ID already exists! Please use another ID.');

```

```

$query = "INSERT INTO `user_table`
(`User_ID`, `User_name`, `User_email`, `User_pwd`, `User_position`, `User_start`)
VALUES (?, ?, ?, ?, ?, ?)";

$stmt = mysqli_prepare($conn, $query);

if ($stmt) {
    mysqli_stmt_bind_param($stmt, "ssssss", $USER_id, $USER_name, $USER_email, $HASHED_pwd, $USER_position, $USER_start);

    if (mysqli_stmt_execute($stmt)) {
        $_SESSION['message'] = "<script>alert('Registered new staff successful. You may ask the staff to login now.');

```

Figure 19: SQL Statement Modifications (Staff Insert)

```

$query = "UPDATE `user_table` SET
    `User_position` = ?
    WHERE `User_ID` = ?";
$stmt = mysqli_prepare($connected, $query);

// Check if the prepared statement is successfully updated
if ($stmt) {
    // Check the number of affected rows after the update operation
    mysqli_stmt_bind_param($stmt, "ss", $POSITION, $ID);

    // Execute the statement
    if (mysqli_stmt_execute($stmt)) {
        $rowsAffected = mysqli_stmt_affected_rows($stmt);

        if ($rowsAffected > 0) {
            $_SESSION['message'] = "<script>alert('Staff's role updated!');</script>";
            header("location:Staff.php?st=updated");
        } else {
            $_SESSION['message'] = "<script>alert('Staff's role update failed. Please try again.');" . "</script>";
            header("location:Staff.php?st=failure");
        }
    }
}

```

Figure 20: SQL Statement Modifications (Staff Update)

```

$query = "DELETE FROM order_table WHERE ID = ?";
$stmt = mysqli_prepare($connected, $query);

// Check if the prepared statement is successfully deleted
if ($stmt) {
    mysqli_stmt_bind_param($stmt, "s", $DELETE_id);

    // Execute the statement
    if (mysqli_stmt_execute($stmt)) {
        // Check the number of affected rows after the delete operation
        $rowsAffected = mysqli_stmt_affected_rows($stmt);

        if ($rowsAffected > 0) {
            header("location:Order_Edit.php?id=$TABLE&st=Cancelled");
        } else {
            $_SESSION['message'] = "<script>alert('Order cancel failed. Please try again.');" . "</script>";
            header("location:Order_Edit.php?id=$TABLE&st=failure");
        }
    } else {
        // echo "Error executing the statement: " . mysqli_stmt_error($stmt);
        $_SESSION['message'] = "<script>alert('Order cancel failed. Please try again.');" . "</script>";
        header("location:Order_Edit.php?st=failure");
    }

    // Close the prepared statement
    mysqli_stmt_close($stmt);
}

```

Figure 21: SQL Statement Modifications (Order Delete)

```
$query = "UPDATE `order_table` SET
        `Food_quantity` = ?
        WHERE `ID` = ?";
$stmt = mysqli_prepare($connected, $query);
```

```
// Check if the prepared statement is successfully updated
if ($stmt) {
    // Check the number of affected rows after the update operation
    mysqli_stmt_bind_param($stmt, "ss", $QUANTITY, $ID);

    // Execute the statement
    if (mysqli_stmt_execute($stmt)) {
        $rowsAffected = mysqli_stmt_affected_rows($stmt);

        if ($rowsAffected > 0) {
            $_SESSION['message'] = "<script>alert('Order updated !');</script>";
            header("location:Order_Edit.php?id=$TABLE");
        } else {
            $_SESSION['message'] = "<script>alert('Order update failed. Please try again.');" . "</script>";
            header("location:Order.php?st=failure");
        }
    } else {
        // echo "Error executing the statement: " . mysqli_stmt_error($stmt);
        $_SESSION['message'] = "<script>alert('Order update failed. Please try again.');" . "</script>";
        header("location:Order.php?st=failure");
    }

    // Close the prepared statement
    mysqli_stmt_close($stmt);
}
```

Figure 22: SQL Statement Modifications (Order Update)

```
$query_insert = "INSERT INTO `order_table` (`Table_ID`, `Food_name`, `Food_quantity`) VALUES (?, ?, ?)";
$stmt_insert = mysqli_prepare($connected, $query_insert);

if ($stmt_insert) {
    mysqli_stmt_bind_param($stmt_insert, "ssi", $TABLE_id, $FOOD_name, $QUANTITY);

    if (mysqli_stmt_execute($stmt_insert)) {
        $query_update = "UPDATE `table_data` SET `Live_status` = 'Seated' WHERE Table_ID = ?";
        $stmt_update = mysqli_prepare($connected, $query_update);

        if ($stmt_update) {
            mysqli_stmt_bind_param($stmt_update, "s", $TABLE_id);

            if (mysqli_stmt_execute($stmt_update)) {
                $_SESSION['message'] = "<script>alert('New food order added!');</script>";
                header("location:Order_Edit.php?id=" . $TABLE_id);
            } else {
                $_SESSION['message'] = "<script>alert('Update table status failed. Please try again.');" . "</script>";
                header("location:Order_Edit.php?st=failure");
            }

            // Close the prepared statement
            mysqli_stmt_close($stmt_update);
        } else {
            $_SESSION['message'] = "<script>alert('Update table status failed. Please try again.');" . "</script>";
            header("location:Order_Edit.php?st=failure");
        }

        // Close the prepared statement
        mysqli_stmt_close($stmt_insert);
    }
}
```

Figure 23: SQL Statement Modifications (Order Insert)

```

$query = "DELETE FROM table_data WHERE Table_ID = ?";
$stmt = mysqli_prepare($connected, $query);

// Check if the prepared statement is successfully deleted
if ($stmt) {
    mysqli_stmt_bind_param($stmt, "s", $DELETE_id);

    // Execute the statement
    if (mysqli_stmt_execute($stmt)) {
        // Check the number of affected rows after the delete operation
        $rowsAffected = mysqli_stmt_affected_rows($stmt);

        if ($rowsAffected > 0) {
            header("location:Table.php?st=Deleted");
        } else {
            $_SESSION['message'] = "<script>alert('Table delete failed. Please try again.');

```

Figure 24: SQL Statement Modifications (Table Delete)

```

$query = "UPDATE `table_data` SET
`Capacity` = ?,
`Table_status` = ?
WHERE `Table_ID` = ?";
$stmt = mysqli_prepare($connected, $query);

// Check if the prepared statement is successfully updated
if ($stmt) {
    // Check the number of affected rows after the update operation
    mysqli_stmt_bind_param($stmt, "iss", $CAPACITY, $STATUS, $ID);

    // Execute the statement
    if (mysqli_stmt_execute($stmt)) {
        $rowsAffected = mysqli_stmt_affected_rows($stmt);

        if ($rowsAffected > 0) {
            header("location:Table.php?st=updated");
            $_SESSION['message'] = "<script>alert('Table capacity updated !');

```

Figure 25: SQL Statement Modifications (Table Update)

```

$query = "INSERT INTO `table_data`
        (`Table_ID`, `Capacity`, `Table_status`)
        VALUES (?, ?, ?)";
$stmt = mysqli_prepare($conn, $query);

if ($stmt) {
    mysqli_stmt_bind_param($stmt, "sds", $TABLE_ID, $TABLE_capacity, $TABLE_status);

    if (mysqli_stmt_execute($stmt)) {
        $_SESSION['message'] = "<script>alert('New table added!');</script>";
        header("location:Table.php?st=success");
    } else {
        $_SESSION['message'] = "<script>alert('Add new table failed. Please try again.');" . "</script>";
        header("location:Table.php?st=failure");
    }
}
// Close the prepared statement
mysqli_stmt_close($stmt);

```

Figure 26: SQL Statement Modifications (Table Insert)

```

$query = "DELETE FROM tax_table WHERE Tax_ID = ?";
$stmt = mysqli_prepare($conn, $query);

// Check if the prepared statement is successfully deleted
if ($stmt) {
    mysqli_stmt_bind_param($stmt, "i", $DELETE_id);

    // Execute the statement
    if (mysqli_stmt_execute($stmt)) {
        // Check the number of affected rows after the delete operation
        $rowsAffected = mysqli_stmt_affected_rows($stmt);

        if ($rowsAffected > 0) {
            header("location:Tax.php?st=Deleted");
        } else {
            $_SESSION['message'] = "<script>alert('Tax delete failed. Please try again.');" . "</script>";
            header("location:Tax.php?st=failure");
        }
    } else {
        // echo "Error executing the statement: " . mysqli_stmt_error($stmt);
        $_SESSION['message'] = "<script>alert('Tax delete failed. Please try again.');" . "</script>";
        header("location:Tax.php?st=failure");
    }
}

// Close the prepared statement
mysqli_stmt_close($stmt);

```

Figure 27: SQL Statement Modifications (Tax Delete)

```

$query = "INSERT INTO `tax_table`
        (`Tax_name`, `Tax_percent`, `Tax_status`)
        VALUES (?, ?, ?)";
$stmt = mysqli_prepare($conn, $query);

if ($stmt) {
    mysqli_stmt_bind_param($stmt, "sss", $TAX_name, $percent, $TAX_status);

    if (mysqli_stmt_execute($stmt)) {
        $_SESSION['message'] = "<script>alert('New tax added!');</script>";
        header("location:Tax.php?st=success");
    } else {
        $_SESSION['message'] = "<script>alert('Add new tax failed. Please try again.');" . "</script>";
        header("location:Tax.php?st=failure");
    }

    // Close the prepared statement
    mysqli_stmt_close($stmt);
}

```

Figure 28: SQL Statement Modifications (Tax Insert)

```

$query = "UPDATE `tax_table` SET
        `Tax_name` = ?,
        `Tax_percent` = ?,
        `Tax_status` = ?
        WHERE `Tax_ID` = ?";
$stmt = mysqli_prepare($conn, $query);

```

```

// Check if the prepared statement is successfully updated
if ($stmt) {
    // Check the number of affected rows after the update operation
    mysqli_stmt_bind_param($stmt, "ssss", $NAME, $percent, $STATUS, $ID);

    // Execute the statement
    if (mysqli_stmt_execute($stmt)) {
        $rowsAffected = mysqli_stmt_affected_rows($stmt);

        if ($rowsAffected > 0) {
            $_SESSION['message'] = "<script>alert('Tax detail updated!');</script>";
            header("location:Tax.php?st=updated");
        } else {
            $_SESSION['message'] = "<script>alert('Tax detail update failed. Please try again.');" . "</script>";
            header("location:Tax.php?st=failure");
        }
    } else {
        // echo "Error executing the statement: " . mysqli_stmt_error($stmt);
        $_SESSION['message'] = "<script>alert('Tax detail update failed. Please try again.');" . "</script>";
        header("location:Tax.php?st=failure");
    }

    // Close the prepared statement
    mysqli_stmt_close($stmt);
}

```

Figure 29: SQL Statement Modifications (Tax Update)


```

$query = "INSERT INTO `waitlist_table`
        (`Wait_ID`, `Cus_name`, `Cus_Pax`, `Cus_contact`, `Wait_time`)
        VALUES (?, ?, ?, ?, ?)";
$stmt = mysqli_prepare($conn, $query);

if ($stmt) {
    mysqli_stmt_bind_param($stmt, "sssss", $WAIT_ID, $CUS_name, $CUS_pax, $CUS_contact, $CUS_time);

    if (mysqli_stmt_execute($stmt)) {
        $_SESSION['message'] = "<script>alert('New waitlist added!');</script>";
        header("location:Waitlist.php?st=success");
    } else {
        $_SESSION['message'] = "<script>alert('Add new waitlist failed. Please try again.');

```

Figure 30: SQL Statement Modifications (Wait Insert)

```

$query = "UPDATE `waitlist_table` SET
        `Cus_name` = ?,
        `Cus_Pax` = ?,
        `Cus_contact` = ?
        WHERE `Wait_ID` = ?";
$stmt = mysqli_prepare($conn, $query);

// Check if the prepared statement is successfully updated
if ($stmt) {
    // Check the number of affected rows after the update operation
    mysqli_stmt_bind_param($stmt, "ssiss", $NAME, $PAX, $CONTACT, $ID);

    // Execute the statement
    if (mysqli_stmt_execute($stmt)) {
        $rowsAffected = mysqli_stmt_affected_rows($stmt);

        if ($rowsAffected > 0) {
            $_SESSION['message'] = "<script>alert('Waitlist updated!');</script>";
            header("location:Waitlist.php?st=updated");
        } else {
            $_SESSION['message'] = "<script>alert('Waitlist update failed. Please try again.');

```

Figure 31: SQL Statement Modifications (Wait Update)

```

$query_assign = "DELETE FROM waitlist_table WHERE Wait_ID = ?";
$statement_assign = mysqli_prepare($connected, $query_assign);

// Check if the prepared statement is successfully deleted
if ($statement_assign) {
    mysqli_stmt_bind_param($statement_assign, "s", $DELETE_id);

    // Execute the statement
    if (mysqli_stmt_execute($statement_assign)) {
        // Check the number of affected rows after the delete operation
        $rowsAffected = mysqli_stmt_affected_rows($statement_assign);

        if ($rowsAffected > 0) {
            header("location:Order_Edit.php?id=$table");
        } else {
            $_SESSION['message'] = "<script>alert('Assign table failed. Please try again.');

```

```

$query = "DELETE FROM waitlist_table WHERE Wait_ID = ?";
$statement = mysqli_prepare($connected, $query);

// Check if the prepared statement is successfully deleted
if ($statement) {
    mysqli_stmt_bind_param($statement, "s", $DELETE_id);

    // Execute the statement
    if (mysqli_stmt_execute($statement)) {
        // Check the number of affected rows after the delete operation
        $rowsAffected = mysqli_stmt_affected_rows($statement);

        if ($rowsAffected > 0) {
            header("location:Waitlist.php?st=Deleted");
        } else {
            $_SESSION['message'] = "<script>alert('Waiting order delete failed. Please try again.');

```

Figure 32: SQL Statement Modifications (Wait Delete)

```

$query = "UPDATE `user_table` SET
        `User_name` = ?,
        `User_email` = ?,
        `User_pwd` = ?,
        `User_contact` = ?,
        `User_birthday` = ?,
        `User_gender` = ?
        WHERE User_ID = ?";
$stmt = mysqli_prepare($connected, $query);

```

```

// Check if the prepared statement is successfully updated
if ($statement) {
    // Check the number of affected rows after the update operation
    mysqli_stmt_bind_param($statement, "ssssss", $NAME, $EMAIL, $HASHEDPWD, $CONTACT, $BIRTHDAY, $ID);

    // Execute the statement
    if (mysqli_stmt_execute($statement)) {
        $rowsAffected = mysqli_stmt_affected_rows($statement);

        if ($rowsAffected > 0) {
            $_SESSION['message'] = "<script>alert('Profile updated!');</script>";
            header("location:Profile.php?st=success");
        } else {
            $_SESSION['message'] = "<script>alert('Profile update failed. Please try again.');"
            header("location:Profile.php?st=failure");
        }
    }
}

```

Figure 33: SQL Statement Modifications (Profile Update)

By employing prepared statements 'mysqli_prepare' to thwart SQL injection, the implementation places a strong emphasis on SQL security. Security is improved by parameter binding and data type specification 'mysqli_stmt_bind_param'. Accurate data processing is ensured by essential data types. "mysqli_stmt_execute" allows for safe and sanitised query execution. 'mysqli_stmt_affected_rows' evaluates operation impact, improving security and control over database operations.

d. Integration of CAPTCHA Codes

```
<?php
include("connect.php");

session_start();

$object = new Connect();

if (!$object->SettedUp()) {
    header("location:" . $object->base_url . "RegisterForm.php");
    exit();
}

// Initialize variables
$isRecaptchaVerified = false;
$recaptchaError = '';

// Check if the form is submitted
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Verify reCAPTCHA
    $recaptchaSecretKey = '6LdGKBgpAAAAANTTlv9EblGac0vE9TK2LTrSXH2h';
    $recaptchaResponse = $_POST['g-recaptcha-response'];

    $recaptchaUrl = 'https://www.google.com/recaptcha/api/siteverify';
    $recaptchaData = [
        'secret' => $recaptchaSecretKey,
        'response' => $recaptchaResponse,
    ];

    $options = [
        'http' => [
            'header' => 'Content-type: application/x-www-form-urlencoded',
            'method' => 'POST',
            'content' => http_build_query($recaptchaData),
        ],
    ];

    $context = stream_context_create($options);
    $recaptchaResult = file_get_contents($recaptchaUrl, false, $context);
    $recaptchaResult = json_decode($recaptchaResult, true);

    if ($recaptchaResult['success']) {
        // reCAPTCHA verification passed
        $isRecaptchaVerified = true;
    } else {
        // reCAPTCHA verification failed
        $recaptchaError = 'reCAPTCHA verification failed. Please try again.';
    }
}
?>
```

Figure 34: reCAPTCHA Integration

This code validates the application setup and checks user reCAPTCHA response using the supplied secret key. If successful, it sets `$isRecaptchaVerified` to true; otherwise, assigns an error to `$recaptchaError`. This enhances security by ensuring form submissions are from actual users, preventing automated scripts.

```
<!-- reCAPTCHA container -->
<div class="recaptcha-container">
  <div class="g-recaptcha" data-sitekey="6LdGKBgpAAAAAMsydD7B6MPDVDHnFF66JP31kovI"></div>
</div>
```

Figure 35: reCAPTCHA Container

The reCAPTCHA container contains the checkbox where user could complete the CAPTCHA.

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Initialize variables
    $isRecaptchaVerified = false;
    $recaptchaError = '';

    // Verify reCAPTCHA
    $recaptchaSecretKey = '6LdGKBgpAAAAANTTlv9EblGac0vE9TK2LTrSXH2h';
    $recaptchaResponse = $_POST['g-recaptcha-response'];

    $recaptchaUrl = 'https://www.google.com/recaptcha/api/siteverify';
    $recaptchaData = [
        'secret' => $recaptchaSecretKey,
        'response' => $recaptchaResponse,
    ];

    $options = [
        'http' => [
            'header' => 'Content-type: application/x-www-form-urlencoded',
            'method' => 'POST',
            'content' => http_build_query($recaptchaData),
        ],
    ];

    $context = stream_context_create($options);
    $recaptchaResult = file_get_contents($recaptchaUrl, false, $context);
    $recaptchaResult = json_decode($recaptchaResult, true);
```

```

if ($recaptchaResult['success']) {
    // reCAPTCHA verification passed
    $isRecaptchaVerified = true;

    // Continue with the login logic
    if (isset($_POST['Sign'])) {
        $USER_id = $_POST['User_ID'];
        $USER_pwd = $_POST['User_pwd'];

        $query = "SELECT * FROM user_table WHERE User_ID = ?";
        $statement = mysqli_prepare($connected, $query);

        if ($statement) {
            mysqli_stmt_bind_param($statement, "s", $USER_id);
            mysqli_stmt_execute($statement);

            $result = mysqli_stmt_get_result($statement);

            // Check if a row exists for the given User_ID
            if ($row = mysqli_fetch_assoc($result)) {
                // Verify the entered password against the hashed password in the database
                if (password_verify($USER_pwd, $row['User_pwd'])) {
                    $_SESSION['User_ID'] = $row['User_ID'];
                    $DATETIME = $object->get_datetime();

                    mysqli_query($connected, "INSERT INTO `attendance_table` (`Staff_ID`, `LoginTime`) VALUES ('".$_SESSION['User_ID']."', '$DATETIME')");
                    header("location:Dashboard.php");
                } else {
                    $_SESSION['message'] = "Login failed. Please check your ID and password.";
                    header("location:index.php?st=failure");
                }
            } else {
                $_SESSION['message'] = "Login failed. User not found.";
                header("location:index.php?st=failure");
            }
        }
    }

    // Close the statement
    mysqli_stmt_close($statement);
}

} else {
    // reCAPTCHA verification failed
    $recaptchaError = 'reCAPTCHA verification failed. Please try again.';
    $_SESSION['message'] = $recaptchaError;
    header("location:index.php");
}
}

```

Figure 36: reCAPTCHA Integration (Login Check)

This PHP script strengthens user authentication by validating reCAPTCHA before login. Secure authentication involves comparing user-supplied credentials with hashed passwords in the database. Successful logins update attendance records and redirect to the dashboard. Failures display error messages and redirect users. The implementation enhances security with reCAPTCHA validation and secure password comparison.

4.0 Discussion

4.1 Key Findings and Issue Resolution

Overview of Key Findings:

During the review and enhancement of the system, several key findings were identified, focusing on SQL statements, password handling, and the implementation of reCAPTCHA. These findings played a crucial role in fortifying the security posture of the system.

Resolved Issues and Improvements Made:

SQL Statement Modification

- ☐ Identified and rectified potential SQL injection vulnerabilities by modifying SQL statements.
- ☐ Implemented parameterized queries to enhance database interaction security.
- ☐ Addressed potential points of unauthorized access through strengthened SQL defences.

Password Hashing

- ☐ Improved the security of user credentials by implementing advanced password hashing techniques.
- ☐ Migrated from conventional hashing to more secure algorithms, enhancing resistance against password-related attacks.
- ☐ Ensured that stored passwords are not susceptible to compromise, even in the event of a data breach.

reCAPTCHA Implementation

- ☐ Integrated reCAPTCHA to mitigate automated login attempts and enhance overall authentication security.
- ☐ Provided an additional layer of defense against malicious bots and potential brute-force attacks.
- ☐ Established a user-friendly approach to differentiating between genuine users and automated scripts during login attempts.

4.2 Continuous Improvement and Maintenance of Security

Continuous Improvement

Regular Security Audits

- ☐ Arrange regular security audits to find and handle new threats.
- ☐ Perform vulnerability assessments to find and fix possible security flaws early on.
- ☐ Stay up to date on industry standards and best practises to make sure security measures meet the most recent requirements.

Employee Training and Awareness

- ☐ Continuously educate staff on evolving security threats and preventive measures.
- ☐ Foster a security-conscious culture within the organization to promote collective responsibility for safeguarding sensitive information.

Real-Time Monitoring and Incident Response

- ☐ Implement real-time monitoring tools to promptly detect and respond to security incidents.
- ☐ Establish an incident response plan, ensuring a swift and effective reaction to any security breaches.
- ☐ Regularly update and rehearse the incident response plan to maintain its effectiveness.

Adaptive Authentication Measures

- ☐ Explore multifactor authentication (MFA) options to add an extra layer of identity verification.
- ☐ Consider implementing adaptive authentication, dynamically adjusting security measures based on risk assessments and user behaviour.

Maintenance of Security

Regular Software Updates

- ☐ Keep all software components, including the web application, database, and server, up to date.
- ☐ Apply security patches promptly to address known vulnerabilities and ensure the latest security features are active.

Data Backup and Recovery

- ☐ Implement a robust data backup strategy to prevent data loss in case of unexpected incidents.
- ☐ Regularly test data recovery procedures to verify their effectiveness and efficiency.

User Account Management

- ☐ Regularly review and audit user accounts to ensure only necessary personnel have access.
- ☐ Disable or remove inactive accounts promptly to minimize potential security risks.

Documentation and Knowledge Sharing

- ☐ Maintain comprehensive documentation on security configurations and procedures.
- ☐ Facilitate knowledge sharing among the development and security teams to promote a collaborative security approach.

5.0 Summary

5.1 Conclusion

In conclusion, the systematic approach to security upgrades, encompassing SQL statement modifications, advanced password hashing, and the integration of reCAPTCHA, marks a pivotal step towards fortifying the system against potential threats. The dedication to resolving identified issues and instituting ongoing security measures reflects our commitment to maintaining a resilient and secure platform.

5.2 Learning Outcome Achieved

Improved Knowledge of Database Security

- ☐ By modifying SQL statements, we have gained a deeper understanding of how to protect databases from common vulnerabilities, particularly SQL injection.

Useful Applications of Advanced Password Hashing

- ☐ The switch to more secure password hashing methods has given us useful knowledge about how to protect user credentials.

Including reCAPTCHA to Boost Authentication

- ☐ In addition to reducing the possibility of automated attacks, the effective integration of reCAPTCHA shows that we can easily integrate sophisticated security features.

Strategies for Constant Improvement

- ☐ Our commitment to adapting to changing security challenges is demonstrated by the implementation of continuous improvement strategies, such as frequent security audits and employee training.

6.0 References

1. *How to Implement a CAPTCHA Solution on Your Website* (n.d.) available from <https://www.a2hosting.com/blog/implement-captcha-solution-website/> [23 November 2023]
2. *How to Prevent SQL Injection Vulnerabilities: How Prepared Statements Work* (n.d.) available from <https://www.securityjourney.com/post/how-to-prevent-sql-injection-vulnerabilities-how-prepared-statements-work> [23 November 2023]
3. *How to Prevent SQL Injection with Prepared Statements | TechTarget* (n.d.) available from <https://www.techtarget.com/searchsecurity/feature/How-to-prevent-SQL-injection-with-prepared-statements> [23 November 2023]
4. *PHP Password Hashing Tutorial (with Examples) - Alex Web Develop* (n.d.) available from <https://alexwebdevelop.com/php-password-hashing/> [23 November 2023]
5. *ReCAPTCHA | Google for Developers* (n.d.) available from <https://developers.google.com/recaptcha> [23 November 2023]
6. *Sql - PHP Hash Password on Login Page - Stack Overflow* (n.d.) available from <https://stackoverflow.com/questions/75901160/php-hash-password-on-login-page> [23 November 2023]

7.0 Appendix

Github Link: <https://github.com/CrazyJoey4/Security>

Turnitin

As the cover page was forgotten to be removed, the plagiarism rate is 15%.

Security_Coursework_2.docx

ORIGINALITY REPORT

15%

SIMILARITY INDEX

13%

INTERNET SOURCES

0%

PUBLICATIONS

14%

STUDENT PAPERS

PRIMARY SOURCES

1

Submitted to INTI Universal Holdings SDM
BHD

Student Paper

6%

2

sathya.in

Internet Source

5%

3

Submitted to Laureate Education Inc.

Student Paper

2%

4

www.coursehero.com

Internet Source

1%

5

github.coventry.ac.uk

Internet Source

1%

6

Submitted to NCC Education

Student Paper

1%

7

Submitted to University of Greenwich

Student Paper

<1%