Web Application Programming

Week 10. Web Application Security(Implementing security mechanisms)

By Elubu Joseph - MSc.IS

Lecturer

Department of Information Technology

Kumi University

Email: josebulinda@gmail.com or jose@kumiuniversity.ac.ug

Agenda

1. Summary of previous lecture
2. Web Application Security(Implementing security mechanisms:
   i. input validation,
   ii. sanitization,
   iii. prepared statements)

Summary of previous lecture

1. Web Application Security(
   i. Introduced web application security
   ii. Looked at Common web vulnerabilities (e.g., XSS, CSRF, SQL injection))
2. Built User Registration System(with each part of code explained)

Implementing security mechanisms- Updated Registration form

Implementing security in web applications is crucial to protect against various threats and vulnerabilities. We will move though a a modified Registration System code as we deal with the implementation of security measures.

The HTML code below creates a registration form with various input fields (for first name, last name, gender, role, username, email, and password) and a "Submit" button. When the user fills out the form and clicks "Submit," the data will be sent to the server script specified in the **action** attribute ("register1.php") for further processing.

Implementing security mechanisms- Updated Registration form

```
    <center><H2>Sign Up</H2><hr>
<form id="commentform" name="reg" method="POST" action="register1.php">
<table>
<tr><td width=100%>First Name:</td>
<td><input name="firstname">  </td>
</tr>
<tr>
<td>Last Name: </td>
<td><input name="lastname"></td>
</tr>
<tr>
<td> <label for="gender">Gender:</label> </td>
<td><select name="gender" id="gender">
    <option value="male">Male</option>
    <option value="female">Female</option>
    </select>
</td>
</tr>
```

Implementing security mechanisms- Updated Registration form

The provided HTML code represents a registration form that allows users to sign up. Here's an explanation of the form's structure and elements:

1. <H2>Sign Up</H2><hr>: This part of the code centers the "Sign Up" heading and adds a horizontal rule (<hr>) below it for visual separation.

2.<form id="commentform" name="reg" method="POST" action="register1.php">: This is the opening <form> tag. It defines the form with the following attributes:

i. **id**: The form's unique identifier, which can be used for JavaScript or CSS styling.

ii. **name**: The name of the form, which can also be used for scripting purposes.

iii. **method**: The HTTP method used to send form data. In this case, it's set to **POST**, which is commonly used for sending data securely.

Implementing security mechanisms- Updated Registration form

```
<tr>
<td><label for="role">Role:</label> </td>
<td><select name="role" id="role">
    <option value="student">Student</option>
    <option value="lecturer">Lecturer</option>
    <option value="ar">Ragistrar</option>
</select></td>
</tr>
<tr><td width=100%>UserName: </td>
<td><input type="text" name="username"></td></tr>
<tr><td width=100%>Email: </td>
<td><input type="email" name="email"></td></tr>
<tr><td>Password:</td>
<td><input type="password" name="password" ></td></tr>
<tr><td>Re-type Pwd:</td>
<td><input type="password" name="re_type_pwd" ></td>
</tr><tr><td></td>
<td><input class="button" type="submit" name="reg" value="Submit"></td>
```

Implementing security mechanisms- Updated Registration form

iv. **action**: The URL to which the form data will be submitted when the user clicks the "Submit" button. In this case, it's set to "register1.php."

3.**<table>**: This element represents an HTML table that is used to structure the form fields. Tables are sometimes used for creating grid-like layouts.

4. Form Fields:

i. **First Name:** An input field for the user's first name. The **<input>** element has a **name** attribute set to "firstname."

ii. **Last Name:** An input field for the user's last name with the **name** attribute set to "lastname."

iii. **Gender:** A dropdown select element for selecting the user's gender. It uses the **<select>** element with the **name** attribute set to "gender" and includes two options: "Male" and "Female."

Implementing security mechanisms- Updated Registration form

iv. **Role**: A dropdown select element for selecting the user's role. It uses the **<select>** element with the **name** attribute set to "role" and includes three options: "Student," "Lecturer," and "Registrar."

v. **Username**: An input field for the user's chosen username with the **name** attribute set to "username." This is where the user will enter their unique username.

vi. **Email**: An input field for the user's email address with the **name** attribute set to "email." It's specifically designed for email input.

Implementing security mechanisms- Updated Registration form

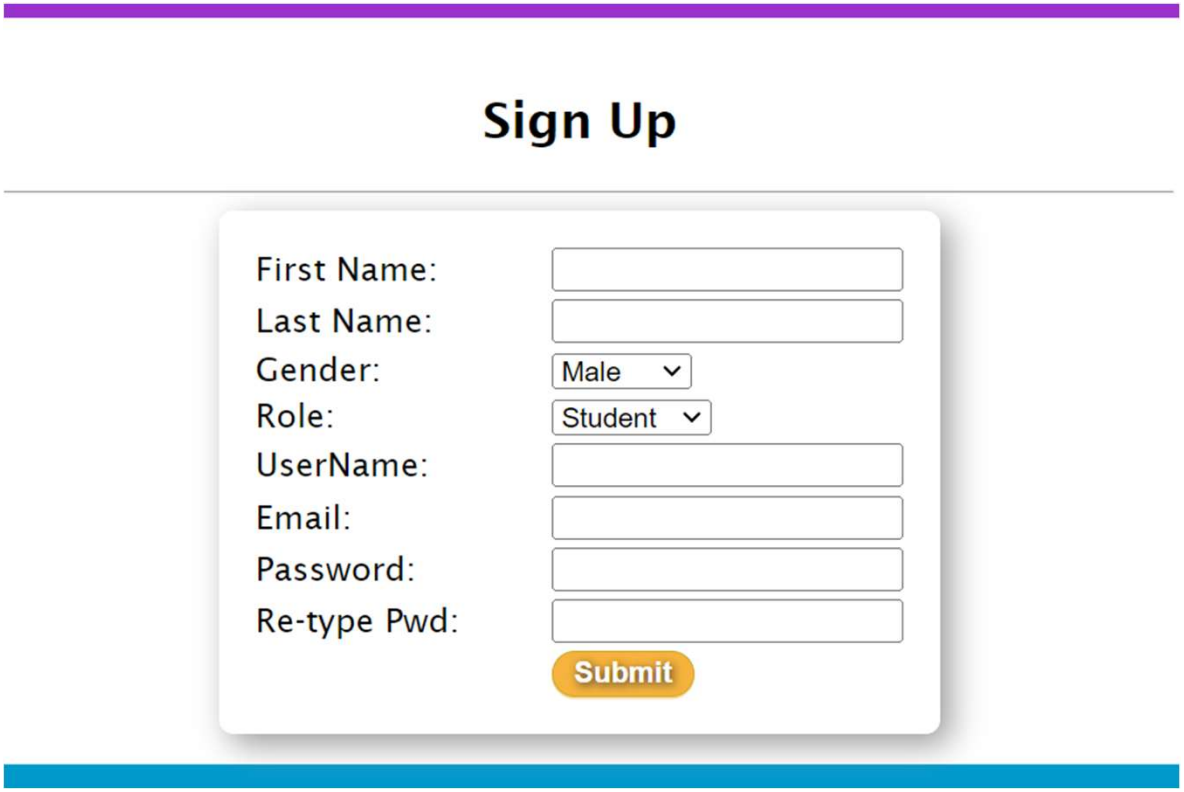
vii. **Password**: An input field for entering the user's password. It uses the **<input>** element with the **type** set to "password" and the **name** attribute set to "password."

viii. **Re-type Password:** Another input field for re-entering the password to confirm it. It's similar to the "Password" field.

ix. Submit Button: A submit button that users can click to submit the form. It has the **type** set to "submit," and its **name** attribute is "reg." The button text is "Submit."

5. **</form>**: This is the closing **</form>** tag, which marks the end of the form.

Implementing security mechanisms- Updated Registration form



Note the presence of two new fields; **gender** and **role**.

Implementing security mechanisms
Updated Table

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	**id**	int(11)			No	*None*		AUTO_INCREMENT	Change Drop More
☐	2	**firstname**	varchar(50)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	3	**lastname**	varchar(50)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	4	**gender**	varchar(6)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	5	**role**	varchar(25)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	6	**username**	varchar(25)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	7	**email**	varchar(80)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	8	**password**	varchar(32)	utf8mb4_general_ci		No	*None*			Change Drop More

Note the increment in the number of fields on the table. Field: role and gender are the new fields on this table.

Web Application Security- Implementing security mechanisms

Input validation

Input validation is an essential part of web application security to prevent various attacks, including SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).

Here's how to implement input validation in PHP.

1.Filter Input Data:

Use the **filter_input** and **filter_var** functions to filter and validate input data. These functions provide a convenient way to validate input against predefined filters.

Web Application Security-
Implementing security mechanisms

Input validation

Here's how to implement input validation in PHP.

2.Sanitize Data:

For each input, apply an appropriate filter to sanitize the data. Sanitization helps remove potentially dangerous or unwanted characters.

3.Check Data Type:

Verify that the input matches the expected data type (e.g., string, integer, email).

Web Application Security-
Implementing security mechanisms

Input validation

Here's how to implement input validation in PHP.

4. **Custom Validation Rules:**

   Create custom validation functions for complex validation rules, such as validating a username or password format.

5. **Validation Error Handling:**

   Check the results of the validation and handle validation errors by displaying appropriate error messages to the user.

Data Validation and sanitization on register1.php

The PHP code is for processing a user registration form. It includes input validation, sanitation, and checks for username and email duplicates. I'll explain the code step by step:

1.**Variable Initialization**: It initializes several variables with empty values. These variables will hold user input data after validation and sanitation.

Implementing security mechanisms- Data Validation & Sanitization

```php
<?php
    require "connect.php";
    require "core.inc.php";

if (isset($_POST['reg'])) {
    // Initialize variables with empty values
    $firstname = $lastname = $gender = $role = $username = $email =
    $password = $password_again = '';

    // Function to sanitize and validate input
    function validateInput($input) {
        return filter_var(trim($input), FILTER_SANITIZE_STRING);
    }
```

2. validateInput Function: A custom function called **validateInput** is defined to sanitize and validate input data using the **FILTER_SANITIZE_STRING** filter. It trims the input and ensures it's a safe string.

Implementing security mechanisms- Data Validation & Sanitization

Focus on FILTER_SANITIZE_STRING filter

The **FILTER_SANITIZE_STRING** filter is one of the filter constants provided by PHP's filter extension. It is used for filtering and sanitizing input data to ensure that it contains only safe, plain text. Here's how it works:

1.**Filter Purpose**: The primary purpose of **FILTER_SANITIZE_STRING** is to remove any characters or elements from the input data that could be considered unsafe, such as HTML tags or special characters.

2.**Usage**: You typically use this filter in combination with the **filter_var()** function to sanitize and validate user input or other data.

Implementing security mechanisms- Data Validation & Sanitization

Focus on FILTER_SANITIZE_STRING filter

3.Sanitization: When **FILTER_SANITIZE_STRING** is applied to the input data, it removes or escapes any potentially harmful characters. This filter is often used to make input safe for display in web applications to prevent Cross-Site Scripting (XSS) vulnerabilities.

Implementing security mechanisms- Data Validation & Sanitization

Focus on FILTER_SANITIZE_STRING filter

Notes: It's important to note that **FILTER_SANITIZE_STRING** focuses on sanitizing data for display and doesn't validate the content of the string.

Overall, the **FILTER_SANITIZE_STRING** filter is a useful tool to sanitize user input and prevent security issues related to unsanitized data. However, it should be used in conjunction with other security measures and tailored to the specific requirements of your application.

Implementing security mechanisms- Data Validation & Sanitization

3. First Name and Last Name Validation: The code checks and sanitizes the first name and last name, ensuring that they are not empty. If they are not empty, the sanitized values are stored in the corresponding variables.

```php
    // Validate and sanitize first name
    if (!empty($_POST['firstname'])) {
        $firstname = validateInput($_POST['firstname']);
    }else {echo "First Name field is empty";}


    // Validate and sanitize last name
    if (!empty($_POST['lastname'])) {
        $lastname = validateInput($_POST['lastname']);
    }else {echo "Last Name field is empty";}
```

Implementing security mechanisms- Data Validation & Sanitization

4. Gender and Role Validation: It validates the selected gender and role by checking if they are not empty and whether they match predefined values ('male', 'female', 'student', 'lecturer', 'ar'). If they match, the values are stored in the corresponding variables.

```php
    // Validate gender
    if (!empty($_POST['gender']) && in_array($_POST['gender'],
        ['male', 'female'])) {
        $gender = $_POST['gender'];
    }

    // Validate role
    if (!empty($_POST['role']) && in_array($_POST['role'],
        ['student', 'lecturer', 'ar'])) {
        $role = $_POST['role'];
    }
```

Implementing security mechanisms- Data Validation & Sanitization

```php
// Validate and sanitize username
if (!empty($_POST['username'])) {
    $username = validateInput($_POST['username']);
}else {echo "UserName field is empty";}

// Validate and sanitize email
if (!empty($_POST['email']) &&
    filter_var($_POST['email'], FILTER_VALIDATE_EMAIL)) {
    $email = validateInput($_POST['email']);
}else {echo "Email field is empty";}
```

6. Username Validation: The code checks the username, ensuring it's not empty. If it's not empty, the validateInput function is applied to sanitize it. If the username is empty, it displays an error message.

7. Email Validation: It validates the email by checking if it's not empty and if it matches a valid email format using the FILTER_VALIDATE_EMAIL filter. If the email is valid, it's sanitized and stored in the email variable.

Implementing security mechanisms- Data Validation & Sanitization

```php
// Validate and sanitize password
if (!empty($_POST['password']) && !empty($_POST['re_type_pwd']) &&
$_POST['password'] === $_POST['re_type_pwd']) {
    $password = password_hash($_POST['password'], PASSWORD_DEFAULT);
}else{echo "password mismatch";}

// Check if the username or email already exists
```

8. Password Validation: It checks if the password and re-typed password match. If they match, it securely hashes the password using the **password_hash()** function with **PASSWORD_DEFAULT**

Implementing security mechanisms- Data Validation & Sanitization

More on PASSWORD_DEFAULT

PASSWORD_DEFAULT is a predefined constant in PHP used as an argument when hashing passwords with the **password_hash()** function. It's designed to automatically select the best and most secure hashing algorithm available on the server at the time the function is called. Here's how it works:

1.**Dynamic Selection**: **PASSWORD_DEFAULT** dynamically selects the most secure hashing algorithm that the PHP version on the server supports. The choice is based on the most secure and recommended algorithm available at that time.

Implementing security mechanisms- Data Validation & Sanitization

More on PASSWORD_DEFAULT

2.**Future-Proofing**: As security evolves and new hashing algorithms are introduced, using **PASSWORD_DEFAULT** ensures that your application will use the best available algorithm without needing to modify your code.

3.**Algorithm Upgrade**: When you use **PASSWORD_DEFAULT** and a new, more secure algorithm becomes available in a future PHP release, your application will automatically benefit from the upgrade without any code changes.

Implementing security mechanisms- Data Validation & Sanitization

More on PASSWORD_DEFAULT

Algorithm Parameters: The **password_hash()** function, when used with **PASSWORD_DEFAULT**, manages the algorithm's parameters internally. It generates a secure salt and applies the recommended number of iterations for password hashing.

Implementing security mechanisms- Data Validation & Sanitization

```php
// Check if the username or email already exists
    $stmt = $conn->prepare("SELECT id FROM user WHERE username = ? OR email = ?");
    $stmt->bind_param("ss", $username, $email);
    $stmt->execute();
    $stmt->store_result();
```

9. Checking for Username or Email Duplicates: The code prepares an SQL statement to check if the username or email already exists in the database. If duplicates are found, it displays an error message. Otherwise, it proceeds to insert the data into the database.

10. Inserting Data into the Database: If there are no duplicates, the code prepares an SQL statement to insert user data into the database. The **bind_param** function is used to bind the sanitized values to the prepared statement.

Implementing security mechanisms- Data Validation & Sanitization

```
    if ($stmt->num_rows > 0) {
            echo "Username or email already exists.
            Please choose a different one.";
    } else {
        // Insert data into the table
        //$password_hash = md5($password);
        $stmt = $conn->prepare("INSERT INTO user (firstname, lastname, gender,
        role, username, email, password) VALUES (?, ?, ?, ?, ?, ?,?)");
        $stmt->bind_param("sssssss", $firstname, $lastname, $username,
        $gender, $role, $email, $password);
```

11. **Inserting Data into the Database**: If there are no duplicates, the code prepares an SQL statement to insert user data into the database. The **bind_param** function is used to bind the sanitized values to the prepared statement.

Implementing security mechanisms- Data Validation & Sanitization

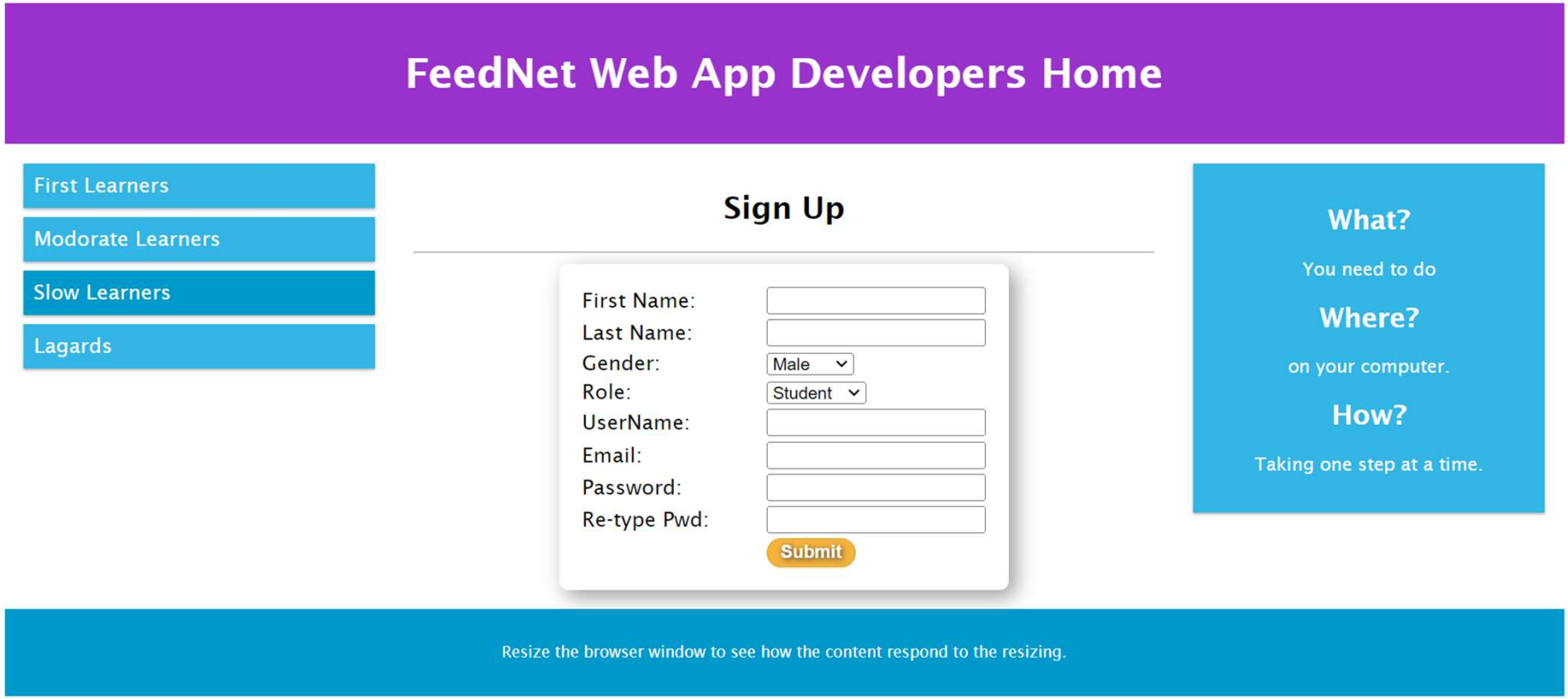
12. **Handling Success and Errors**: If the database insertion is successful, it redirects the user to the index page. If there's an error during database operations, it displays an error message.

13. **Closing Resources**: The code closes the prepared statement and the database connection.

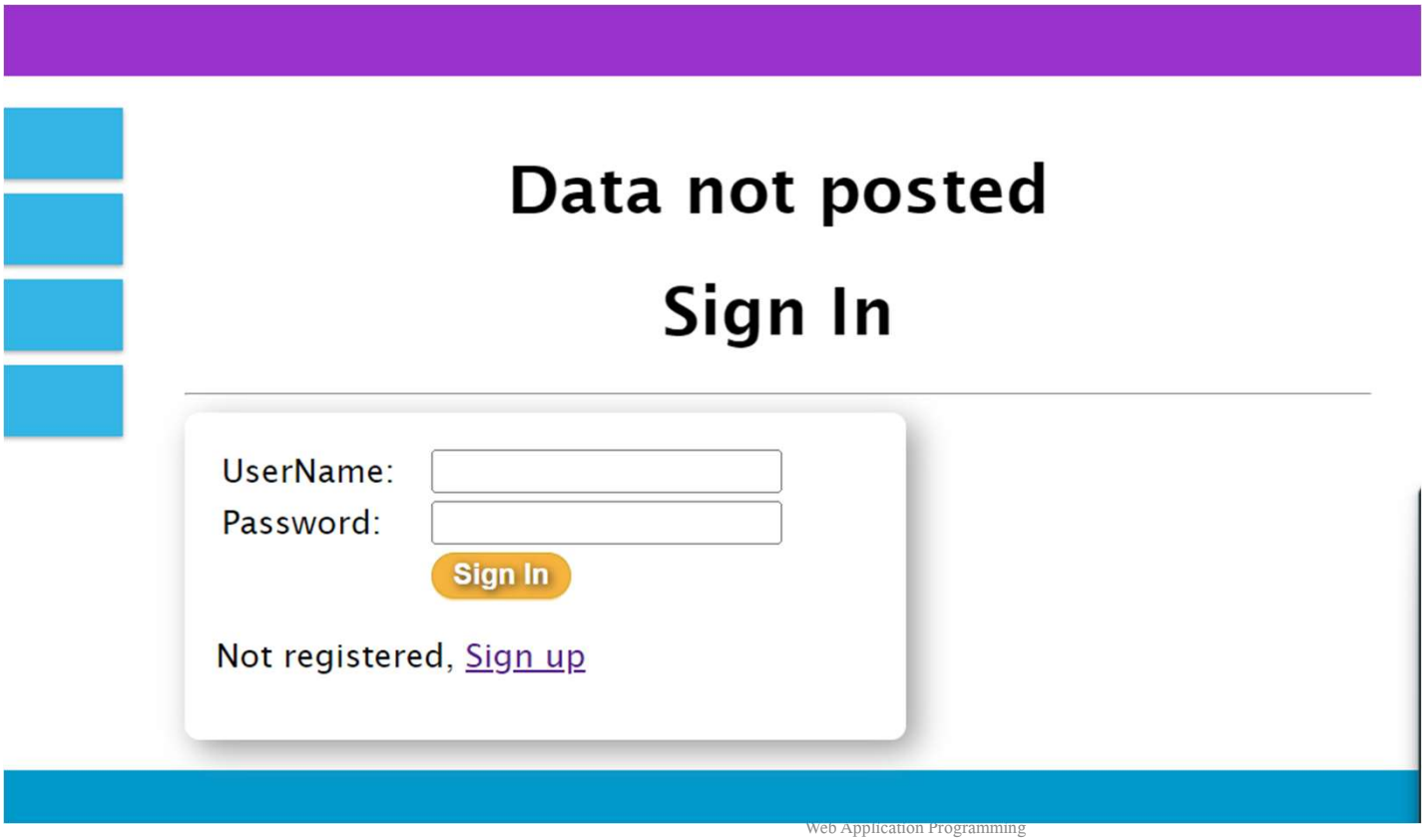
```
        if ($stmt->execute()) {
                header("Location: index.php");
        } else {
                echo "Error: " . $stmt->error;
        }
        }
        $stmt->close();

    $conn->close();
}
```

Implementing security mechanisms- Data Validation & Sanitization



User registration page.



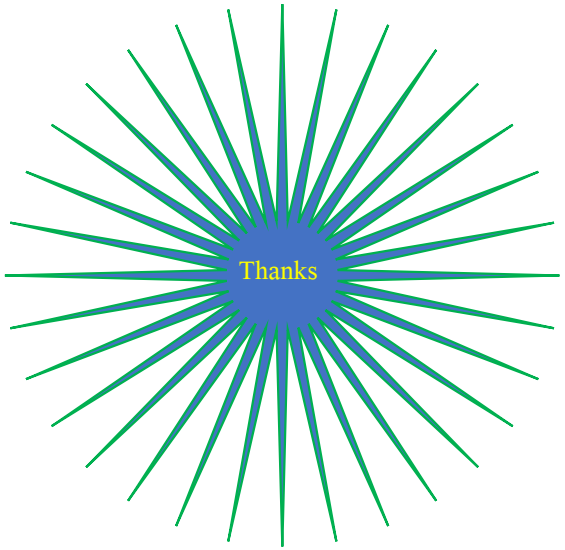
On successful registration, the user is redirected to index.php where loign form is located.

Summary

Summary

1. Web Application Security(Implementing security mechanisms:
   i. input validation(Definition, looked at measures, created **validateInput Function ()** to help with validation, **PASSWORD_DEFAULT** dynamically selects the most secure hashing algorithm),
   ii. Sanitization(used **FILTER_SANITIZE_STRING**),
   iii. prepared statements-)

Thank you for Listening



References

JavaScript Data Types. OpenAI Knowledge Base. OpenAI. (2021, September 22)..