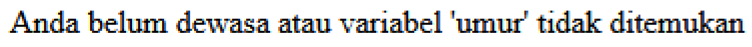
Name: Antonius Kaharap Kautsar
Class: 1-I
NIM: 2341720067

Assignment:

Practical section 1:

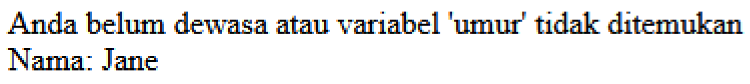
Question 1



```
Anda belum dewasa atau variabel 'umur' tidak ditemukan
```

The code uses isset() to check if the variable $umur exists and if it's greater than or equal to 18. If both conditions are true, it means the person is an adult. If $umur isn't set or is less than 18, it indicates the person is either not an adult or the age information is missing. Essentially, isset() helps avoid errors by ensuring the code only checks the age when the variable actually holds a value.

Question 2

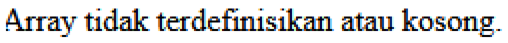


```
Anda belum dewasa atau variabel 'umur' tidak ditemukan
Nama: Jane
```

In this code, isset($data["nama"]) checks if the "nama" key exists within the $data array. If it exists, the code displays the corresponding name ("Jane"). If the "nama" key is not found, it prints a message indicating that the variable is missing from the array. This use of isset() ensures that the code doesn't try to access a non-existent array key, preventing potential errors.

Practical section 2:

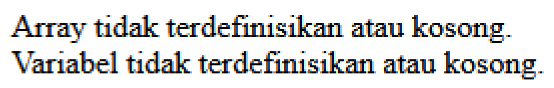
Question 3



```
Array tidak terdefinisikan atau kosong.
```

This code defines an empty array called $myArray. The empty() function checks if this array is indeed empty or not. If $myArray has no elements, the code outputs "Array tidak terdefinisi atau kosong" (Array is undefined or empty). Otherwise, it confirms that the array is defined and contains elements. Essentially, empty() helps determine if an array is empty or if it holds any values.

Question 4

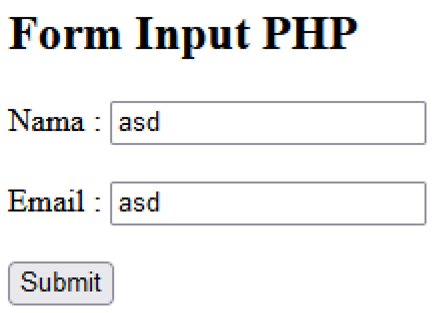


```
Array tidak terdefinisikan atau kosong.
Variabel tidak terdefinisikan atau kosong.
```

In this code, empty($nonExistentVar) checks if a variable named $nonExistentVar is empty. Since this variable hasn't been defined previously, it's considered empty. Therefore, the code outputs "Variabel tidak terdefinisi atau kosong" (Variable is undefined or empty).

Practical section 3:

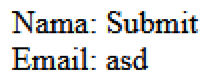
Question 5



Form Input PHP

Nama : asd

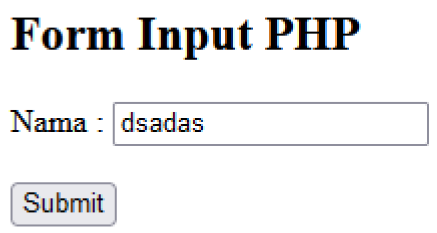
Email : asd



```
Nama: Submit
Email: asd
```

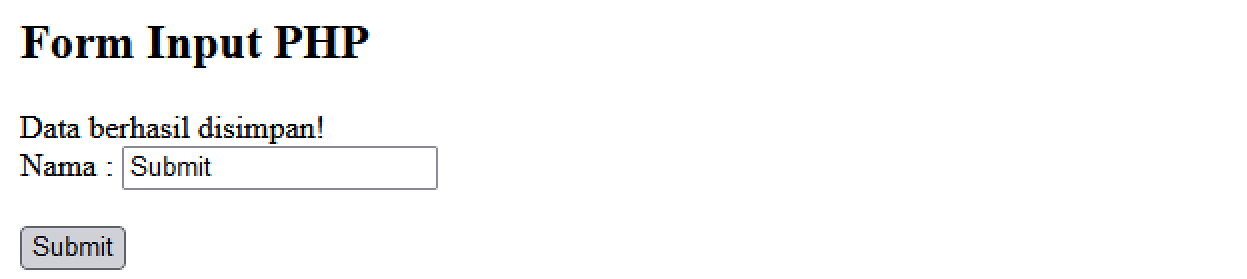
In this code, it will creates a simple form with fields for "name" and "email" and submits the data to proses_form.php. The proses_form.php file checks if the form was submitted via POST, retrieves the values of "name" and "email," and displays them. When you run localhost/week7/proses_form.php without submitting the form, it will not show any output. Running localhost/week7/form.php allows you to input data, and upon submission, the values will be displayed by proses_form.php.

Question 6



Form Input PHP

Nama : dsadas


Form Input PHP

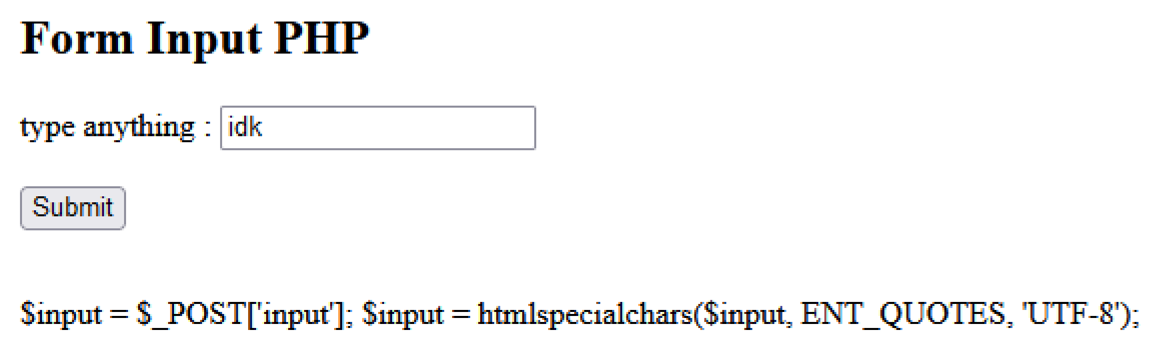
Data berhasil disimpan!

Nama : Submit

In the form_self.php file, the form submits data to the same page using PHP_SELF. The script checks if the form was submitted via POST and validates that the "name" field is not empty. If it's empty, an error message is displayed; if valid, it stores the name and shows a confirmation message. Running localhost/week7/form_self.php allows me to enter a name, and if we submit an empty field, it warns us that the name is required. This process ensures that the form is validated on the same page before submitting the data.

Practical section 4:

Question 7

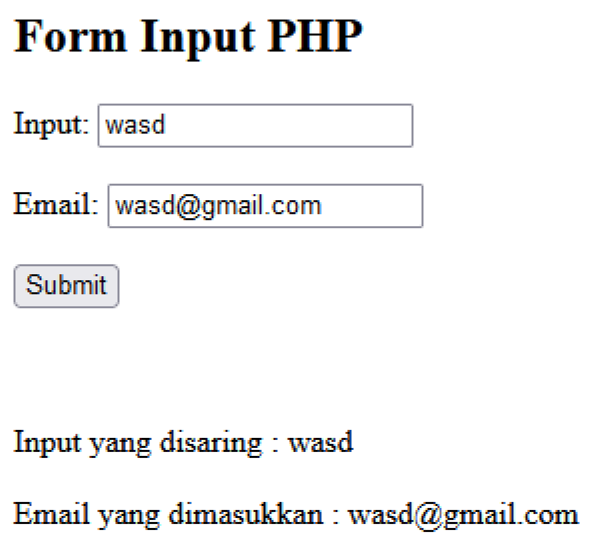

Form Input PHP

type anything : idk

```
$input = $_POST['input']; $input = htmlspecialchars($input, ENT_QUOTES, 'UTF-8');
```

This code demonstrates how failing to sanitize user input can make a website vulnerable to Cross-Site Scripting (XSS) attacks. By removing the htmlspecialchars function, any HTML code entered into the form is rendered directly on the page. This allows attackers to inject malicious scripts, like JavaScript code, that can execute in other users' browsers when they view the page, potentially compromising their data or accounts.

Question 8



Form Input PHP

Input: wasd

Email: wasd@gmail.com

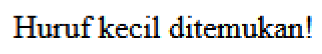
```
Input yang disaring : wasd

Email yang dimasukkan : wasd@gmail.com
```

This code demonstrates input validation using PHP's filter_var function. By specifying FILTER_VALIDATE_EMAIL, the code checks if the submitted input matches the format of a valid email address. If the input is valid, it's processed (in this case, simply displayed); otherwise, an error message is shown. This ensures that only properly formatted email addresses are accepted, preventing potential issues with data storage or sending emails.

Practical section 5:

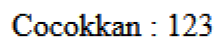
Question 9



```
Huruf kecil ditemukan!
```

This code uses a regular expression (/[a-z]/) to search for lowercase letters within a string. preg_match checks if the pattern matches any part of the text. Since the text "This is a Sample Text." contains lowercase letters, the code outputs "Huruf kecil ditemukan!", indicating a match was found.

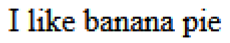
Question 10



```
Cocokkan : 123
```

This code uses the regular expression /[0-9]+/ to find one or more consecutive digits in the string "There are 123 apples." The preg_match function not only checks for a match but also stores the matched value in the $matches array. The code then prints "Cocokkan: 123" because it successfully found the numeric sequence "123" within the text, demonstrating how to extract specific parts of a string using regular expressions.

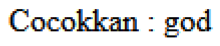
Question 11



```
I like banana pie
```

This code demonstrates string replacement using regular expressions. The preg_replace function searches for the pattern 'apple' within the string "I like apple pie." and replaces it with 'banana'. The output is then "I like banana pie.", showcasing how this function can easily modify text based on patterns, which is useful for tasks like replacing words, correcting typos, or updating information within a string.

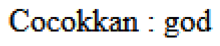
Question 12



```
Cocokkan : god
```

This code uses the regular expression /go*d/ to match variations of the word "good" with any number of "o" characters. The asterisk (*) in the pattern allows for zero or more occurrences of the preceding character. Since the text "god is good." contains both "god" (zero "o"s) and "good" (one "o"), the code outputs "Cocokkan: god" because it finds the first match.

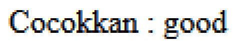
Question 13



```
Cocokkan : god
```

By changing the asterisk (*) to a question mark (?) in the regular expression, the pattern now matches "god" or "good" only. The question mark makes the preceding character ("o") optional, allowing for zero or one occurrence. Therefore, the code outputs "Cocokkan: god" as it finds the first match, which has zero "o"s.

Question 14

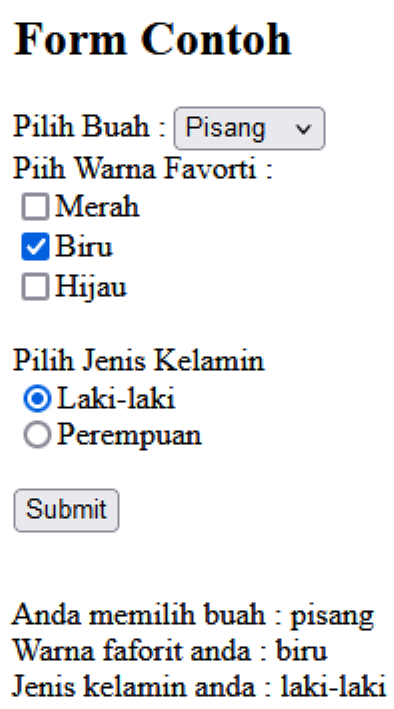


```
Cocokkan : good
```

Changing the pattern to /[o]{1,3}/ now specifically looks for sequences of one to three "o" characters. In the text "god is good.", this matches the "oo" in "good." The code outputs "Cocokkan: oo" because it finds this sequence.

Practical section 6:

Question 15



Form Contoh

Pilih Buah : Pisang

Piih Warna Favorti :

☐ Merah
☑ Biru
☐ Hijau

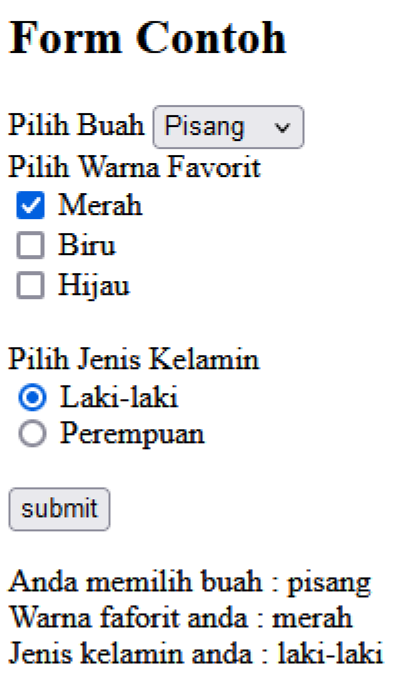
Pilih Jenis Kelamin

[✓ Laki-laki]
Perempuan

```
Anda memilih buah : pisang
Warna faforit anda : biru
Jenis kelamin anda : laki-laki
```

This code creates a basic HTML form (form_lanjut.php) that collects user input for fruit preference, favorite colors (multiple choices possible), and gender. When submitted, the PHP code (proses_lanjut.php) processes this data, displaying the selected fruit and gender. If colors are chosen, it lists them; otherwise, it indicates no selection. This demonstrates handling different input types (dropdown, checkboxes, radio buttons) and processing the data accordingly in PHP.

Question 16



Form Contoh

Pilih Buah Pisang

Pilih Warna Favorit

[✓ Merah]
Biru
Hijau

Pilih Jenis Kelamin

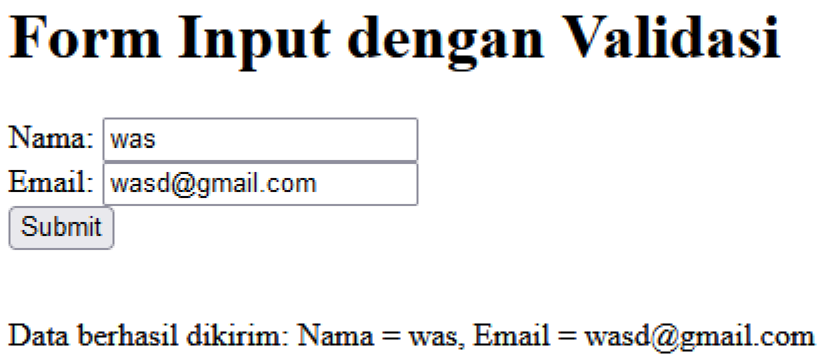
[✓ Laki-laki]
Perempuan

submit

Anda memilih buah : pisang
Warna faforit anda : merah
Jenis kelamin anda : laki-laki

In the form_ajax.php file, the form uses jQuery to submit data asynchronously (via AJAX) to the server without refreshing the page. When I run localhost/week7/form_ajax.php, the form collects the selected fruit, favorite colors, and gender, then sends this data to the server (proses_lanjut.php) using a POST request. The result is displayed in the #hasil div. This approach enhances the user experience by handling form submission dynamically, without needing a full page reload.

Question 17



Form Input dengan Validasi

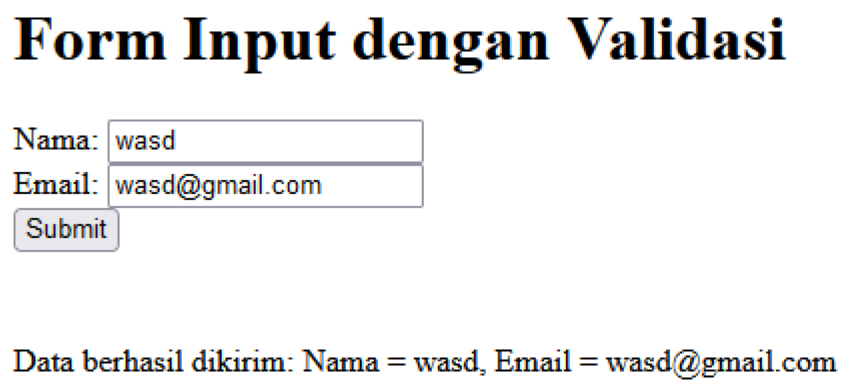
Nama: was
Email: wasd@gmail.com

Submit

Data berhasil dikirim: Nama = was, Email = wasd@gmail.com

This code demonstrates form validation in PHP. The proses_validasi.php script checks if the submitted name and email fields are empty or if the email format is invalid. If errors are found, it displays them; otherwise, it confirms successful submission.

Question 18



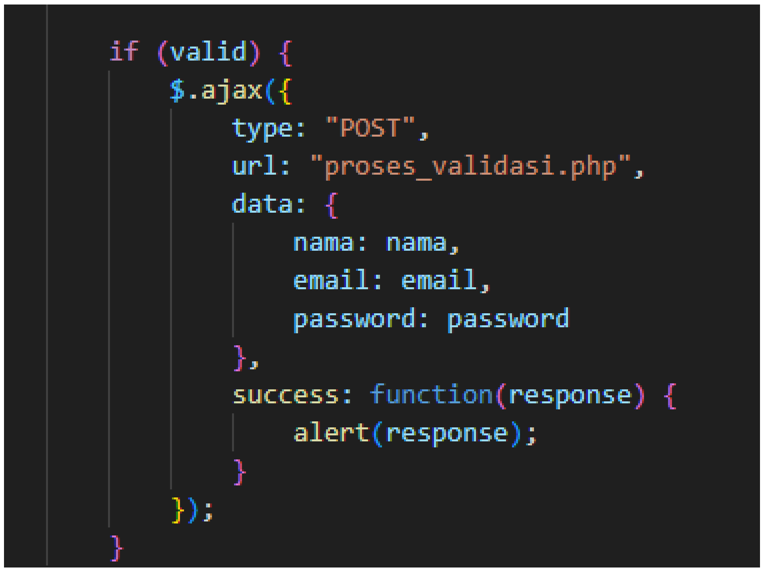
Form Input dengan Validasi

Nama: wasd
Email: wasd@gmail.com
Submit

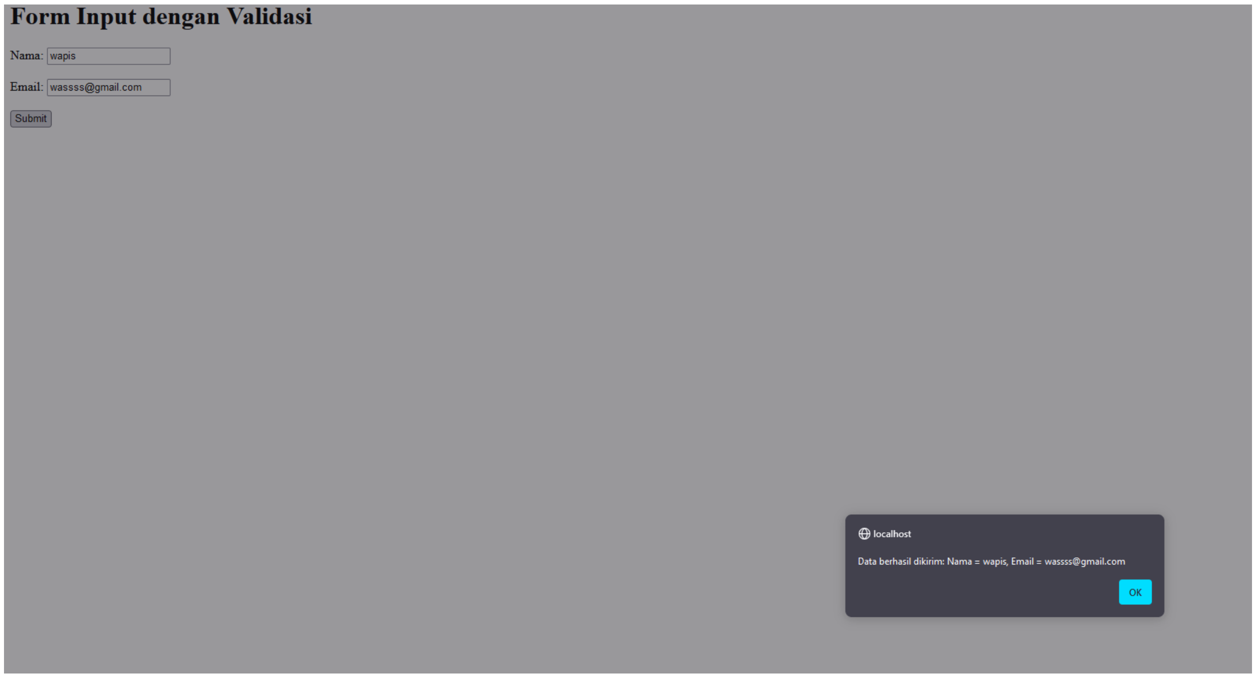
Data berhasil dikirim: Nama = wasd, Email = wasd@gmail.com

This code adds client-side validation to the form using JavaScript and jQuery. Before submitting the form, it checks if the "nama" and "email" fields are empty. If either is empty, it displays an error message next to the field and prevents the form from submitting. This provides immediate feedback to the user, improving the user experience by catching errors before involving the server.

Question 19



```
if (valid) {
    $.ajax({
        type: "POST",
        url: "proses_validasi.php",
        data: {
            nama: nama,
            email: email,
            password: password
        },
        success: function(response) {
            alert(response);
        }
    });
}
```



Form Input dengan Validasi

Nama: wapis

Email: wassss@gmail.com

Submit

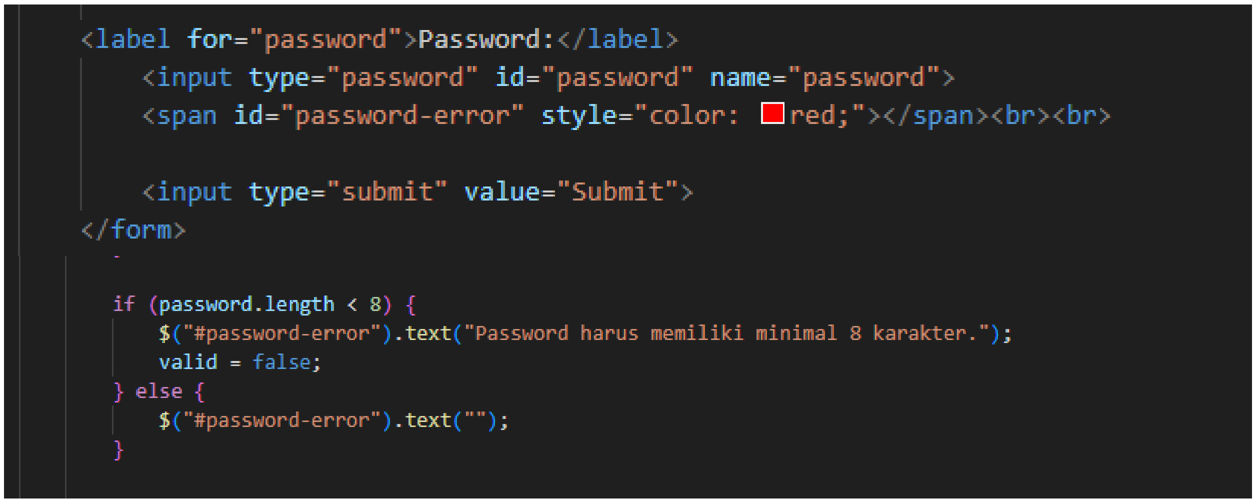
localhost

Data berhasil dikirim: Nama = wapis, Email = wassss@gmail.com

OK

By adding this AJAX script, the form now submits the data to "proses_validasi.php" without reloading the page. The server's response, which contains either error messages or a success message, is then displayed within the form itself (presumably in a designated element with the ID "form-message").

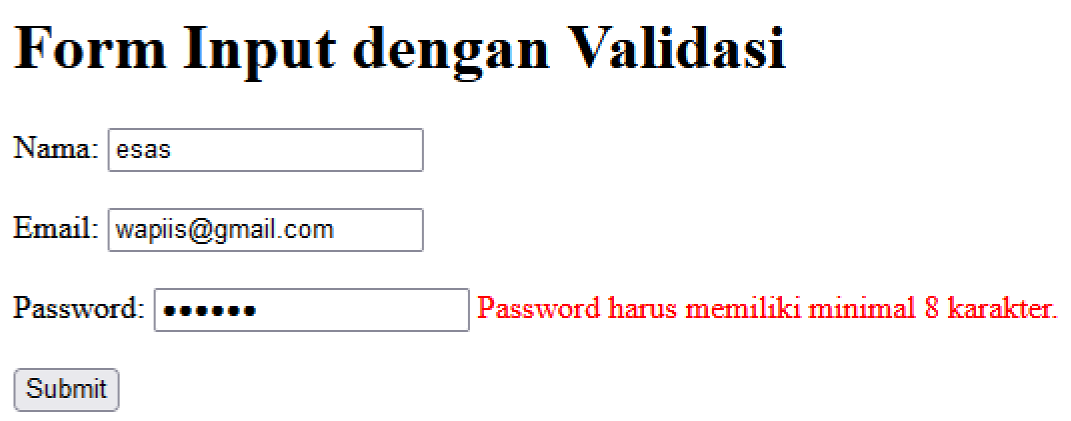
Question 20



```
<label for="password">Password:</label>
    <input type="password" id="password" name="password">
    <span id="password-error" style="color: 🟥red;"></span><br><br>

    <input type="submit" value="Submit">
</form>

  if (password.length < 8) {
      $("#password-error").text("Password harus memiliki minimal 8 karakter.");
      valid = false;
  } else {
      $("#password-error").text("");
  }
```

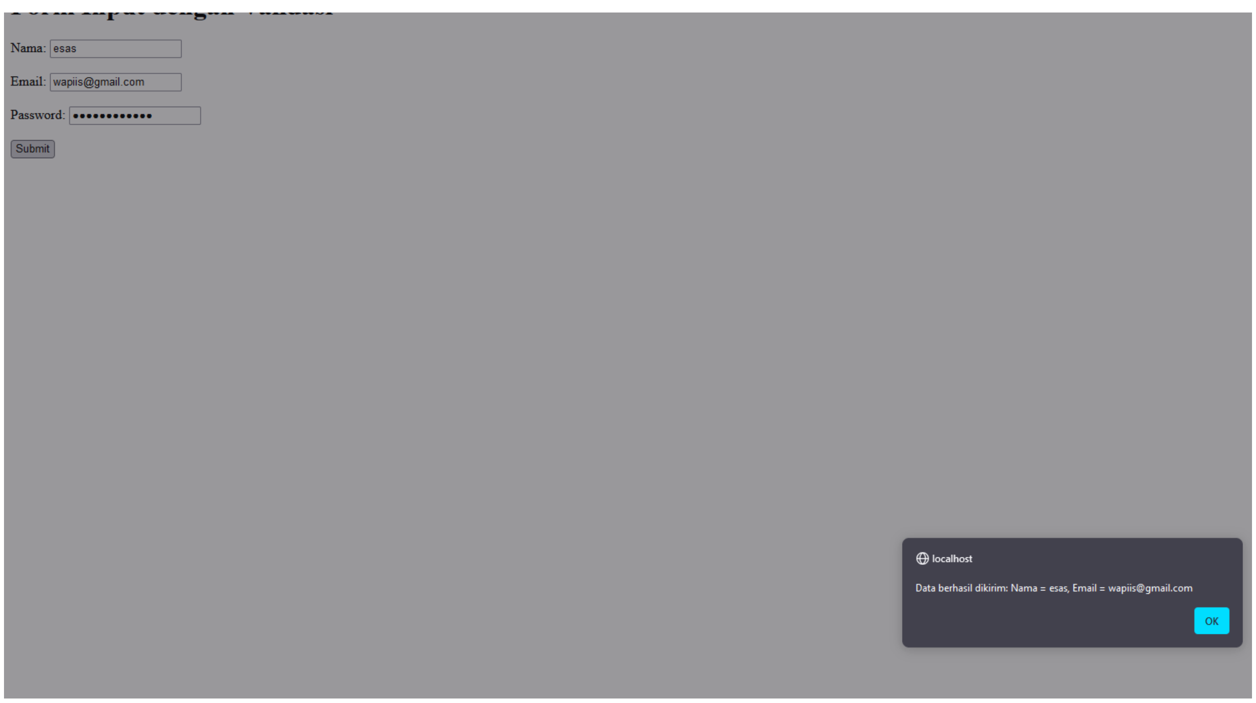


Form Input dengan Validasi

Nama: esas

Email: wapiis@gmail.com

Password: •••••• Password harus memiliki minimal 8 karakter.



Nama: esas

Email: wapiis@gmail.com

Password: ••••••••••••

Data berhasil dikirim: Nama = esas, Email = wapiis@gmail.com

Adding the code on the image enforces password validation on both the client-side (using jQuery) and server-side (using PHP). It checks if the password entered is less than 8 characters long. If it is, an error message is displayed, and the form submission is prevented, ensuring that only passwords meeting the length requirement are accepted.

Github link: https://github.com/Valtern/week-7