WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Shivani Eli

Email: elisi@mail.uc.edu

Figure 1: Shivani Eli

Hackathon 1: Cross-Site Scripting Attacks and Defenses

Overview: In this hackthon, have learnt and done hands-on, covered topics like XSS attacks and defenses, OWASP rules, the vulenrabilities in the code. It has two tasks; Task 1 is all about cross scripting attacks from level 0 to level 6. In task 2 defense, for this user field validations and the output sanitisation were done for mitigating the XSS attacks. This code and report was done in README.md file and converted to pdf file using pandoc tool. this readme was saved under labs/Hackthon-1 directory

https://github.com/ShivaniEli/waph-elisi/blob/main/labs/Hackthon1/README.md

Task 1 : ATTACKS

Level 0

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level0/echo.php

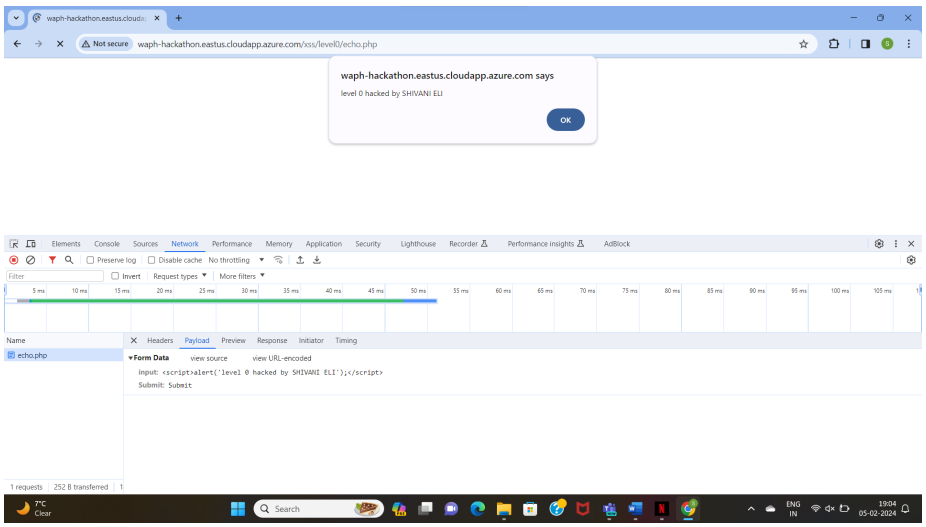


Figure 2: XSS attack level 0

Level 1

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level1/echo.php

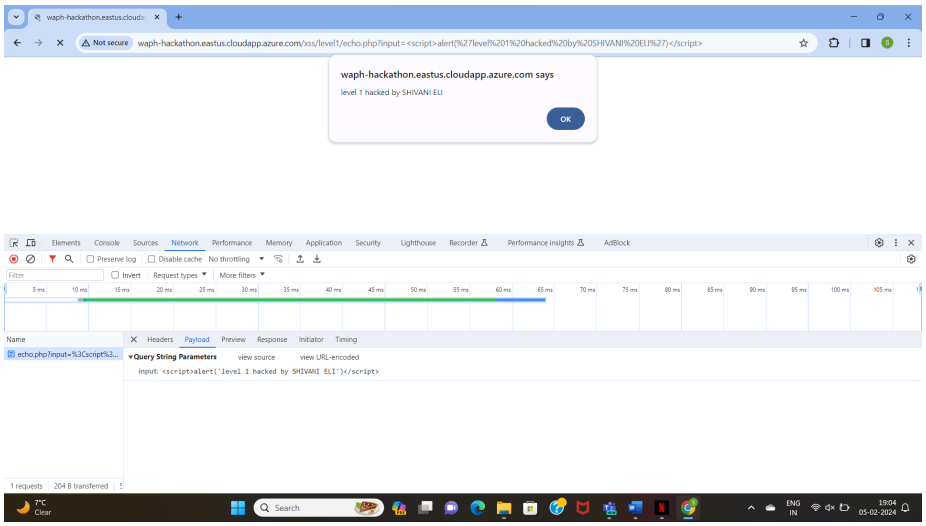


Figure 3: XSS attack level 1

Level 2

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level2/echo.php

guessed source code echo.php:

```
if(!isset($_POST['input'])){
   die("{\"error\": \"Please provide 'input' field in an HTTP POST Request\"}");
echo $_POST['input'];
```

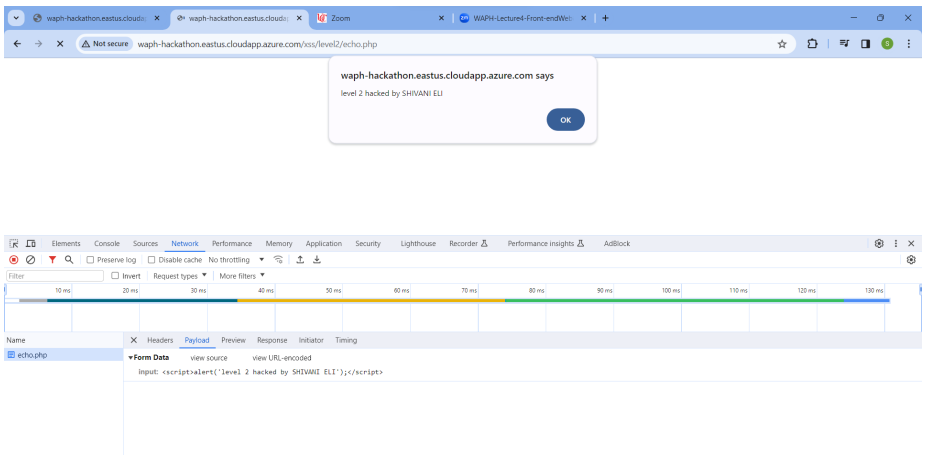


Figure 4: XSS attack level 2

Level 3

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level3/echo.php

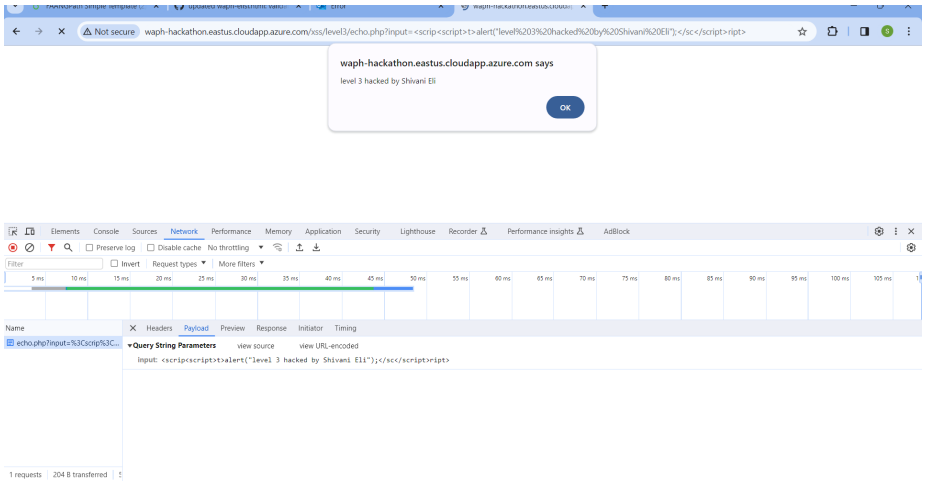


Figure 5: XSS attack level 3

included echo.php guessed source code:

```
str_replace(['<script>', '</script>'], '', $input)
```

Level 4

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level4/echo.php

included guessed code echo.php:

```
$data = $_GET['input']
if (preg_match('/<script\b[^>]*>(.*?)<\/script>/is', $data)) {
    exit('{"error": "No \'script\' is allowed!"}');
    }
else
    echo($data);
```

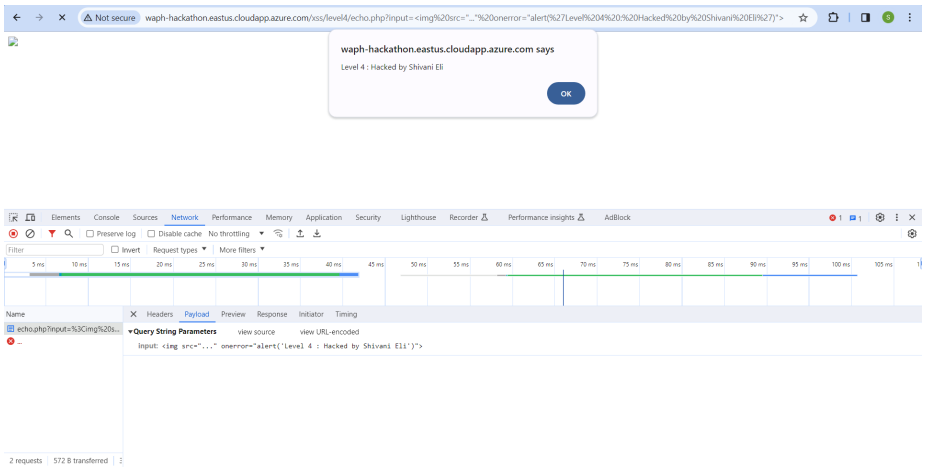


Figure 6: XSS attack level 4

Level 5

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level5/echo.php

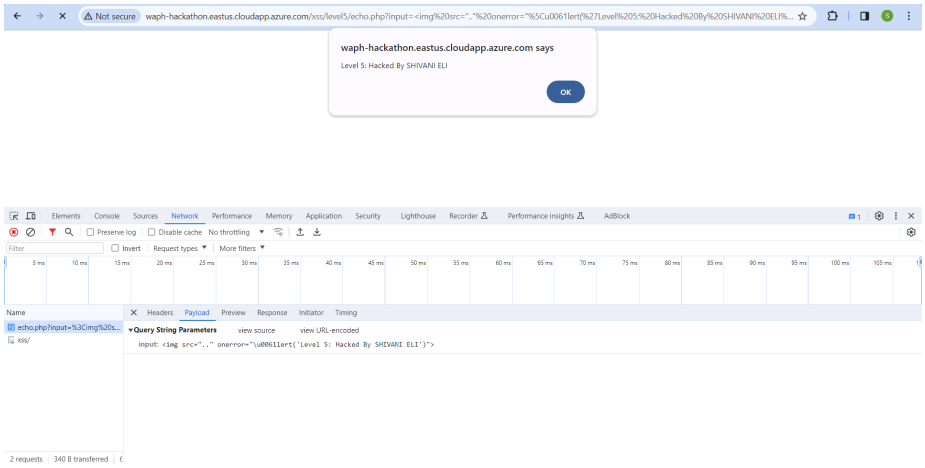


Figure 7: XSS attack level 5

included guessed:

```
$data = $_GET['input']
if (preg_match('/<script\b[^>]*>(.*?)<\/script>/is', $data)
             || stripos($data, 'alert') !== false) {
    exit('{"error": "No \'script\' is allowed!"}');
    }
else
    echo($data);
```

Level 6

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level6/echo.php

included guess code for echo.php :

```
<form action="/xss/level6/echo.php/"
    onkeyup="alert('Level 6 : Hacked by Shivani Eli')" method="POST">
  Input:<input type="text" name="input" />
  <input type="submit" name="Submit"/>
```

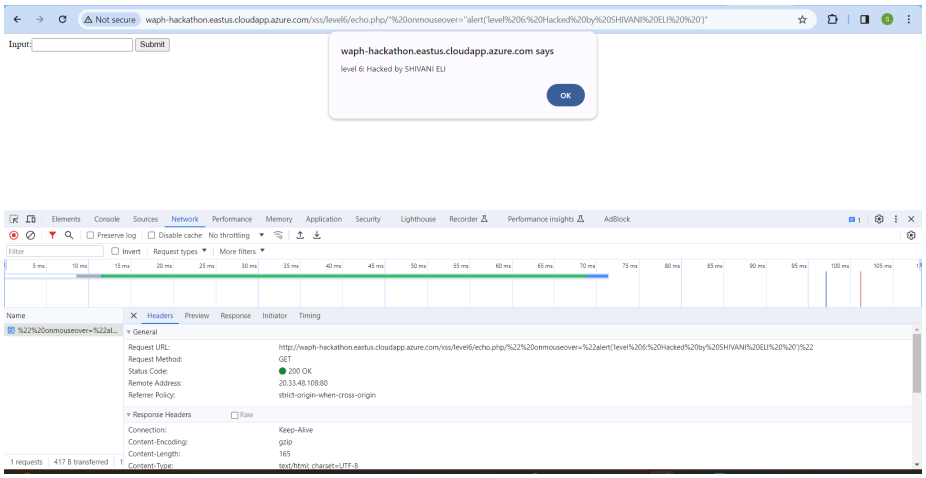


Figure 8: XSS attack level 6

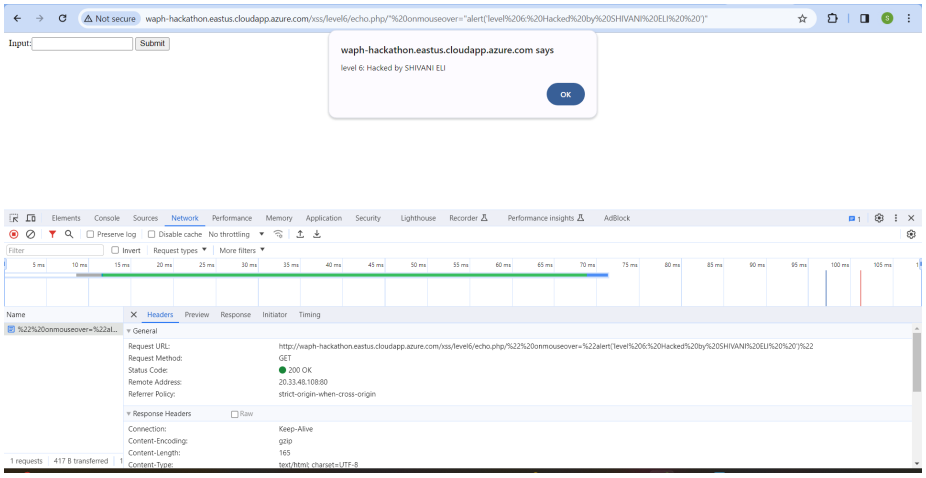


Figure 9: XSS attack level 6 after injecting XSS script code

source code guess of echo.php:

```
echo htmlentities($_REQUEST('input'));
```

TASK 2 : DEFENSE

A . echo.php

In the task2, for the defense, the echo.php file is updated, by adding the user validation in the input field. Then the inputs are verified for empty inputs, then the inputs when valid are checked with function htmlentities().

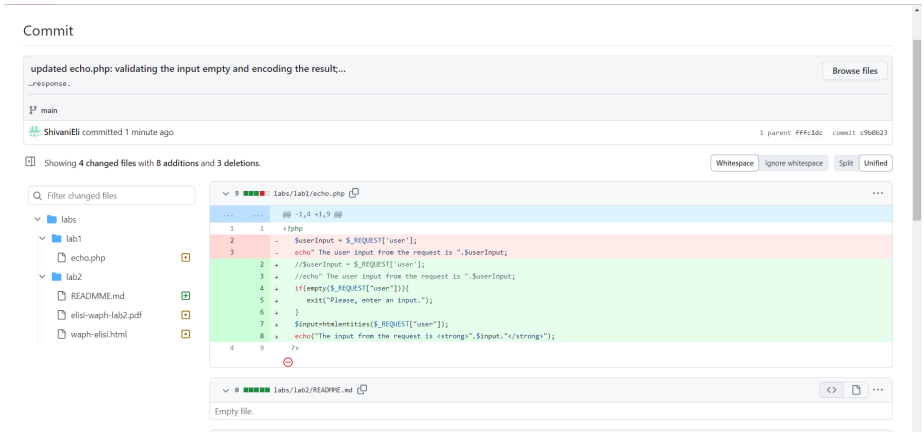


Figure 10: Defense XSS echo.php file

B . Lab 2 front-end part

i) The user validation is done by defining validateInput() function> this is used to validate the user inputs in HTTP requests in the forms. where the users need to enter their input then the requests are executed.

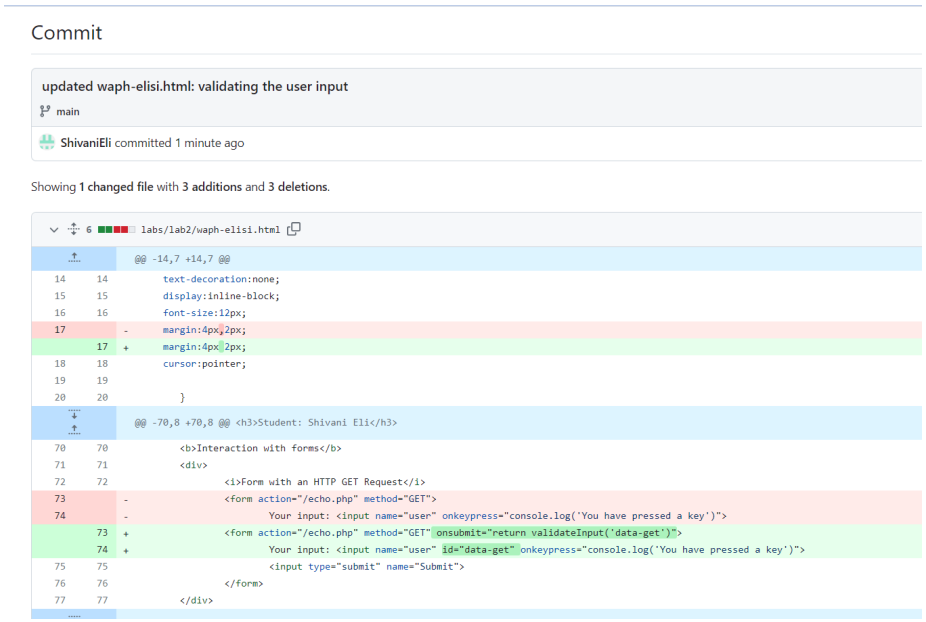


Figure 11: Defense waph-elisi.html

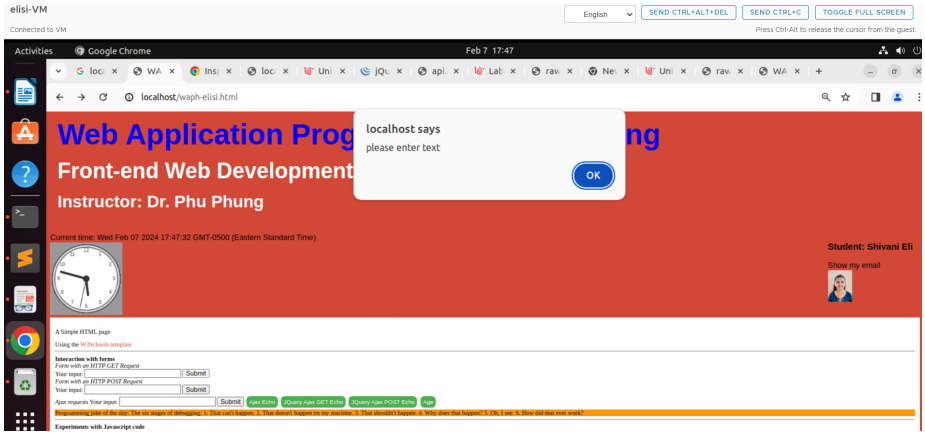


Figure 12: Validating HTTP requests input

ii) In this part, the `.innerHTML` was replaced to `.innerText` so only text is allowed and that the HTML rendering is not done.

```
147  170   }
148  171           </script>
149  172           <hr>
150  173           <b>Experiments with Javascript code</b><br>
151  174           <i>Inlined Javascript</i>
152       -        <div id="date" onclick="document.getElementById('date').innerHTML=Date()">Click here to show Data()</div>
     175  +        <div id="date" onclick="document.getElementById('date').innerText=Date()">Click here to show Data()</div>
153  176   </div>
154  177
155  178   </body>
```

```
147  170   }
148  171           </script>
149  172           <hr>
150  173           <b>Experiments with Javascript code</b><br>
151  174           <i>Inlined Javascript</i>
152       -        <div id="date" onclick="document.getElementById('date').innerHTML=Date()">Click here to show Data()</div>
     175  +        <div id="date" onclick="document.getElementById('date').innerText=Date()">Click here to show Data()</div>
153  176   </div>
154  177
155  178   </body>
```

iii) In this defense, the special characters are converted to HTML entities for prevention of the XSS attacks, by defining the encodeInput() function, returns the data in the div element.

```
function encodeInput(input){
            const encodeData = document.createElement('div');
            encodeData.innerText=input;
            return encodeData.innerHTML;
        }
```

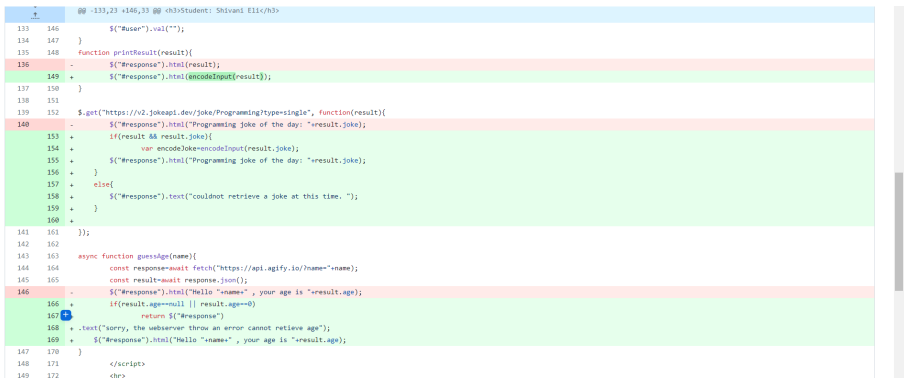
Showing **1 changed file** with **30 additions** and **7 deletions**.

Whitespace | Ignore whitespace | Split | Unified

```diff
@@ -45,6 +45,19 @@ <h3>Student: Shivani Eli</h3>
 45  45             document.getElementById('digital-clock').innerHTML = "Current time: "+new Date();
 46  46         }
 47  47         setInterval(displayTime, 500);
+    48         function validateInput(inputId){
+    49             var input=document.getElementById(inputId).value;
+    50             if(input.length==0){
+    51                 alert("please enter text");
+    52                 return false;
+    53             }
+    54             return true;
+    55         }
+    56         function encodeInput(input){
+    57             const encodeData=document.createElement('div');
+    58             encodeData.innerText=input;
+    59             return encodeData.innerHTML;
+    60         }
 48  61     </script>
 49  62     <canvas id="analog-clock" width="150" height="150" style="background-color:#999 ;"></canvas>
 50  63     <script src="https://waph-uc.github.io/clock.js"></script>
@@ -77,8 +90,8 @@ <h3>Student: Shivani Eli</h3>
 77  90         </div>
 78  91         <div>
 79  92             <i>Form with an HTTP POST Request</i>
-80     -           <form action="/echo.php" method="POST" name="echo_post">
-81     -               Your input: <input name="user">
+    93 +           <form action="/echo.php" method="POST" name="echo_post" onsubmit="return validateInput('data-post')">
+    94 +               Your input: <input name="user" id="data-post" onkeypress="console.log('you have clicked a key')">
 82  95             <input type="submit" name="Submit">
 83  96             </form>
 84  97         </div>
@@ -103,7 +116,7 @@ <h3>Student: Shivani Eli</h3>
103 116         xhttp.onreadystatechange=function(){
104 117             if(this.readyState==4 && this.status==200){
105 118                 console.log("Recieved data="+xhttp.responseText);
106    -                document.getElementById("response").innerText="Response from server:"+ xhttp.responseText;
    119 +                document.getElementById("response").innerText=encodeInput(xhttp.responseText);
107 120             }
108 121         }
109 122         xhttp.open("GET","echo.php?user= "+input, true);
@@ -133,23 +146,33 @@ <h3>Student: Shivani Eli</h3>
133 146         $("#user").val("");
134 147     }
135 148     function printResult(result){
136    -        $("#response").html(result);
    149 +        $("#response").html(encodeInput(result));
137 150     }
```

iv) Another modification that was done is that in the api jokes, code was changed by adding the validation for the joke being loaded, for result.joke is JSON not empty.

```
if (result && result.joke) {
        var encodeJoke = encodeInput(result.joke);
        $("#response").text("Programming joke of the day: " +encodeJoke);
                }
else{
         $("#response").text("couldnot retriev a joke at this time.");
}
```



```
@@ -133,23 +146,33 @@ <h3>Student: Shivani Eli</h3>
133 146          $("#user").val("");
134 147  }
135 148  function printResult(result){
136     -        $("#response").html(result);
    149 +        $("#response").html(encodeInput(result));
137 150  }
138 151
139 152  $.get("https://v2.jokeapi.dev/joke/Programming?type=single", function(result){
140     -        $("#response").html("Programming joke of the day: "+result.joke);
    153 +        if(result && result.joke){
    154 +                var encodeJoke=encodeInput(result.joke);
    155 +        $("#response").html("Programming joke of the day: "+result.joke);
    156 +    }
    157 +    else{
    158 +        $("#response").text("couldnot retrieve a joke at this time. ");
    159 +    }
    160 +
141 161  });
142 162
143 163  async function guessAge(name){
144 164          const response=await fetch("https://api.agify.io/?name="+name);
145 165          const result=await response.json();
146     -        $("#response").html("Hello "+name+" , your age is "+result.age);
    166 +        if(result.age==null || result.age==0)
    167 +                return $("#response")
    168 + .text("sorry, the webserver throw an error cannot retieve age");
    169 +    $("#response").html("Hello "+name+" , your age is "+result.age);
147 170  }
148 171          </script>
149 172          <hr>
```

Figure 13: the modification in joke api function and guessAge()

v) The guessAge() was modified for validating the input from users, it is made to ensure that the input is valid, this enables the user to not allow empty or invalid inputs.

```
if(result.age==null || result.age==0)
   return $("#response")
   .text("sorry, the webserver throw a error cannot retrieve age");
$("#response").text("Hello "+name+" ,your age should be "+result.age);
```

Figure 14: guessAge() function output for invalid input

**********END**********