



**SOFE 2720: Principles of Software and Requirements**  
**Project Deliverable 3**  
**Group 25**

Name	Student Number
Yusuf Shaik	100 655 451
Hamza Farhat Ali	100 657 374
Ala Ghazal Aswad	100 278 910

**Table of Contents**

1.Github Link	2
2.User Stories Implemented	2
3. Project Planning (Also refer to the Issue tab on Github)	4
4. Test Cases	5
4.1 Unit Testing	5
4.2 Integration Testing	11
4.3 Acceptance Testing	13

## 1.Github Link

<https://github.com/Yusuf-shaik/Sudoku>

## 2.User Stories Implemented

User Stories	Requirements
As a user I create an account so that I may play the game	<ul style="list-style-type: none"><li>• User creates an account with their email and password</li><li>• Developers implement sessions so that each user has their own account</li><li>• User logs into account to play game</li></ul>
As a user I want a grid to be displayed so that I can see the game and interact with the game board	<ul style="list-style-type: none"><li>• System renders a 9x9 grid, made up of 9 3x3 grids</li><li>• System displays 3 options for 3 different difficulty levels</li><li>• System displays some values in the grid already filled in, so the user can complete the puzzle. More values are given for an easier levels, and less values are given for a higher difficulty level</li><li>• User is redirected to the login page if the information does not match an account to allow user to retry</li></ul>
As user I want the functionality of sudoku to be available to me so that I can play a real sudoku game	<ul style="list-style-type: none"><li>• System allows only the numbers from 1-9 to be displayed exactly once in each row</li><li>• System allows only the numbers from 1-9 to be displayed exactly once in each column</li><li>• System allows only the numbers from 1-9 to be displayed exactly once in each 3x3 sub-grid</li><li>• Webpage displays an alert when a column, row, or 3x3 sub-grid is full, and the numbers do NOT include the values from 1-9 exactly once each</li></ul>

As a user I want to be able to submit my puzzle after I am finished with it so that my information can be saved.	<ul style="list-style-type: none"> <li>● System renders a submit button at the bottom of the page</li> <li>● Submit button checks to see if the puzzle is correct.</li> <li>● If the puzzle is correct, the game is completed and the score is saved</li> <li>● If the puzzle is incorrect, an alert is displayed to the user telling them that it's wrong</li> </ul>
As a user I want to be able to save my progress before signing out so that I can continue playing another time	<ul style="list-style-type: none"> <li>● System renders a "Save Progress" button for membership users</li> <li>● Clicking the button saves the users information to the database</li> </ul>
As a user I want a reset button so that if I have found an easier way to complete the puzzle, or I have made too many mistakes, I can restart the same puzzle with empty values	<ul style="list-style-type: none"> <li>● System renders a "Reset" button</li> <li>● Clicking the reset button clears all user-inputted values from the screen</li> <li>● User is given an option to reset timer or to continue with the current timer</li> </ul>
As a user I want to be able to try a new puzzle so that I can try a new puzzle if I get stuck	<ul style="list-style-type: none"> <li>● System renders a "New Puzzle" button for membership users</li> <li>● Clicking the button generates a new puzzle for the user to complete with the same difficulty</li> <li>● The timer resets</li> <li>● All progress on the previous game is lost</li> </ul>
As a user I want to be able to see how long I have been playing for so that I can track my progress and see how I am improving at the game	<ul style="list-style-type: none"> <li>● System calculates time and displays it after the game is submitted correctly.</li> </ul>
As a user i want customizable features that will enhance my aesthetics when playing.	<ul style="list-style-type: none"> <li>● User can choose between different genres of music while playing. The feature to hear in the background or hide the music playing is also available.</li> <li>● User can also change the background colour according to their choice.</li> </ul>

### 3. Project Planning (Also refer to the Issue tab on Github)

People	<p>The communication for this project was mostly done over messenger and then group meetings.</p> <ol style="list-style-type: none"><li>1. Yusuf and Hamza- Collaborated to implement the the user stories into the Sudoku game. Yusuf worked on the functionality of the game as well as the extra features that were added such as music, background change, reset button etc. Hamza checked the validation of the sign up and login pages for the user. Also added the correct test cases to report. Both team mates implemented bootstrap as the framework along the process.</li><li>2. Ala- was in charge of the report, made sure that all the work was completed.</li></ol>
Product	<p>The product that was created is a web application for a Souko game.</p>
Process	<ol style="list-style-type: none"><li>1. An agile process was used to encompass the software engineering tasks to publish this web application.</li><li>2. Bootstrap was the only framework that was used in order to create this web application.</li></ol>
Project	<p>Implementation of the software developed involved group meeting. This was done to communicate and plan the splitting of work and tasks completed. Finally the construction and deployment of the application was done through github.</p>

## 4. Test Cases

### 4.1 Unit Testing

Test units #	Test Case Description	Test Data	Expected result
1	Check response when user attempts to resets	Click on reset button	Game has successfully reset
2	Checks user input under username, password, credit card and email	The user inputs are left blank or are not of valid length and characters	Alert box appears with appropriate warning.
3	Check response when valid email and password are entered	Email: <a href="mailto:ABSI@email.com">ABSI@email.com</a> Password: 56429A	Login is successful
4	Check to see if same puzzles will show upon completion of first one	Solve the puzzle successfully	Different puzzle will show up for different difficulty
5	Check to see if a timer would reset if a reset button was clicked	Reset button in clicked	Empty puzzle is displayed

The snippets of each code unit is added with descriptions of the functionality. As well as the

```
//check solution for medium game
function say(){
    var x=document.forms["form"];

    for (var i = 0; i <solution.length; i++) {
        userInput.push(x.elements[i].value);

        if (userInput[i]==solution[i]) {
            sum++;
        }
        else
        {
            sum+=0;
            // console.log(userInput[i])
        }
    }
    if (sum==81) {
        alert("You Won!")
    }
    else{
        alert("You Lost!")
    }

    end();
}
```

Figure 1- This javascript function checks to see if the solution matches the sudoku puzzle for the medium difficulty level. The function is repeated for all difficulty levels, however the id called is different every time. As well as, the solution array the user input array is compared to is different for each level. The output is an alert saying "you won" if the user wins, and an alert saying "you lost" if the user loses. This was tested, and it works successfully in the program.

```

//calculate and print end time for timer
function end(){
    endTime=new Date();

    var totalTime=(endTime-startTime)/1000

    console.log(totalTime);
    // document.getElementById("time").innerHTML= "Total Time: " + Math.floor(totalTime) + " seconds" ;
    if (totalTime<60) {
        alert( "Total Time: " + Math.floor(totalTime) + " seconds" );

    }
    else if (totalTime>=60) {
        minutes=totalTime/60;
        seconds=totalTime%60;
        if (Math.floor(minutes)==1) {
            Msuffix= " minute "
        }
        else{
            Msuffix=" minutes "
        }

        if (Math.floor(seconds==1)) {
            SSuffix=" second "
        }
        else{
            SSuffix = " seconds "
        }
        alert( "Total Time: " + Math.floor(minutes) + Msuffix + Math.floor(seconds) + SSuffix)

    }

    startTime=new Date();

```

Figure 2- The function is added to for the time to show in the alert box when the user clicks submit. The time is converted to minutes if the user takes more than 59 seconds. This implementation is achieved the



```
//reset the hard gameboard
function resetH(){
    if (confirm("Are you Sure you want to reset the game? You will lose all progress?")==true){
        document.getElementById("formHard").reset();
    }
}
}
```

Figure 3- This function clears the sudoku puzzle according to the difficulty level. Which is implemented using a confirmation. The output is an alert asking the user if they want to reset the game board, if the user selects yes, the game will be reset, if the user selects no, the game will not be reset. This was tested and both outputs function successfully

```
// hide music or show music based on users click
var count=0;
function showMusic(){
    var show=document.getElementById("audio");

    if (count%2==1) {
        show.style.display="none"
    }
    if (count%2==0) {
        show.style.display="block"
    }
    count++
}
}
```

Figure 4- The showMusic() function allows the user to hide or display their youtube music while still listening to it. A variable is initialised before the function, the first click allows the youtube block to display the music and the music videos are hidden when the user clicks the button again. The output should be the music showing every second click. This was tested, and it works successfully.

```
//change the background color based on user input
function changeBackground(){
    var color=document.getElementById("colorMe").value
    console.log(color)
    document.body.style.backgroundColor=color
```

Figure 5- This allows the colour palette to display when the button is click on by the user. This is implemented into the game. The output is a the color that the user selected as the background. This was tested, and it works successfully.

```
function userValidation(){
    var name = document.getElementById("userName").value;
    var pass = document.getElementById("password").value;
    var userEmail = document.getElementById("email").value;
    var creditCard = document.getElementById("Card").value;

    var creditCardLength=creditCard.toString().length
    // console.log(thename)

    var x=true
    var y=true
    var z=true
    var w=true
    var v=true

    if (name=="") //validates that there are no digits in name
    {
        alert("Please enter a username, it cannot be blank");
        z=false
    }

    // // if (confirmation != thename)
    // // {
    // //     alert("Please make sure usernames match");
    // //     return false;
    // // }

    if (pass == "")
    {
        alert("please Enter a Password");
        w=false
    }

    if (userEmail== "")
    {
        alert("Please enter an email");
        x=false
    }
}
```

Figure 6- This function allows the validation of the input fields of all name, password, credit card etc. This is by checking that they are not empty and are implemented on the web application. The output should be an alert each time a condition is not satisfied. Each case was tested, and they all function successfully.

## 4.2 Integration Testing

```
//check solution for easy game
function sayEasy(){
    var x=document.forms["formEasy"];

    for (var i = 0; i <easySolution.length; i++) {
        userInput.push(x.elements[i].value);

        if (userInput[i]==easySolution[i]) {
            sum++;
        }
        else
        {
            sum+=0;
            // console.log(userInput[i])
        }
    }
    if (sum%81==0) {
        alert("You Won!")
    }
    else{
        alert("You Lost!")
    }
    console.log(sum);

    end();
}
```

Figure 7- This function checks the answer that the user inputted by pushing the values to an array, and comparing the array with a solution array. If all the values match, the answer will be "you won" and if any of the values don't match, the output will be "you lost". This function is the submit function of the game, after we check the answer, we will need to calculate the time of the game. This is done by calling another function, the "end()" function shown in figure 8

```

function end(){
    endTime=new Date();

    var totalTime=(endTime-startTime)/1000

    console.log(totalTime);
    // document.getElementById("time").innerHTML= "Total Time: " + Math.floor(totalTime) + " seconds" ;
    if (totalTime<60) {
        alert( "Total Time: " + Math.floor(totalTime) + " seconds" );
    }
    else if (totalTime>=60) {
        minutes=totalTime/60;
        seconds=totalTime%60;
        if (Math.floor(minutes)==1) {
            Msuffix= " minute "
        }
        else{
            Msuffix=" minutes "
        }

        if (Math.floor(seconds==1)) {
            SSuffix=" second "
        }
        else{
            SSuffix = " seconds "
        }
        alert( "Total Time: " + Math.floor(minutes) + Msuffix + Math.floor(seconds) + SSuffix)
    }

    startTime=new Date();
}

```

Figure 8- This function calculates the time it takes for the user to finish the game. The output will be the time that the page has been loaded for until the user clicked submit. It will be in minutes if the time is longer than 60 seconds. The function will work whether the user has won or lost their game.

This shows the integration of the sayEasy() function with the end() function as it tracks the time when the user solves the puzzle. This is implemented with an alert function in the app. These functions work together. This was tested and all possible outputs work successfully.

Other Functions: The other functions do not have integrations as they are not dependant on each other for the outputs. Other functions like `userValidation()`, `showMusic()`, they have all the required variables to be implemented. It does not need any calls from other functions or to other functions to work.

### 4.3 Acceptance Testing

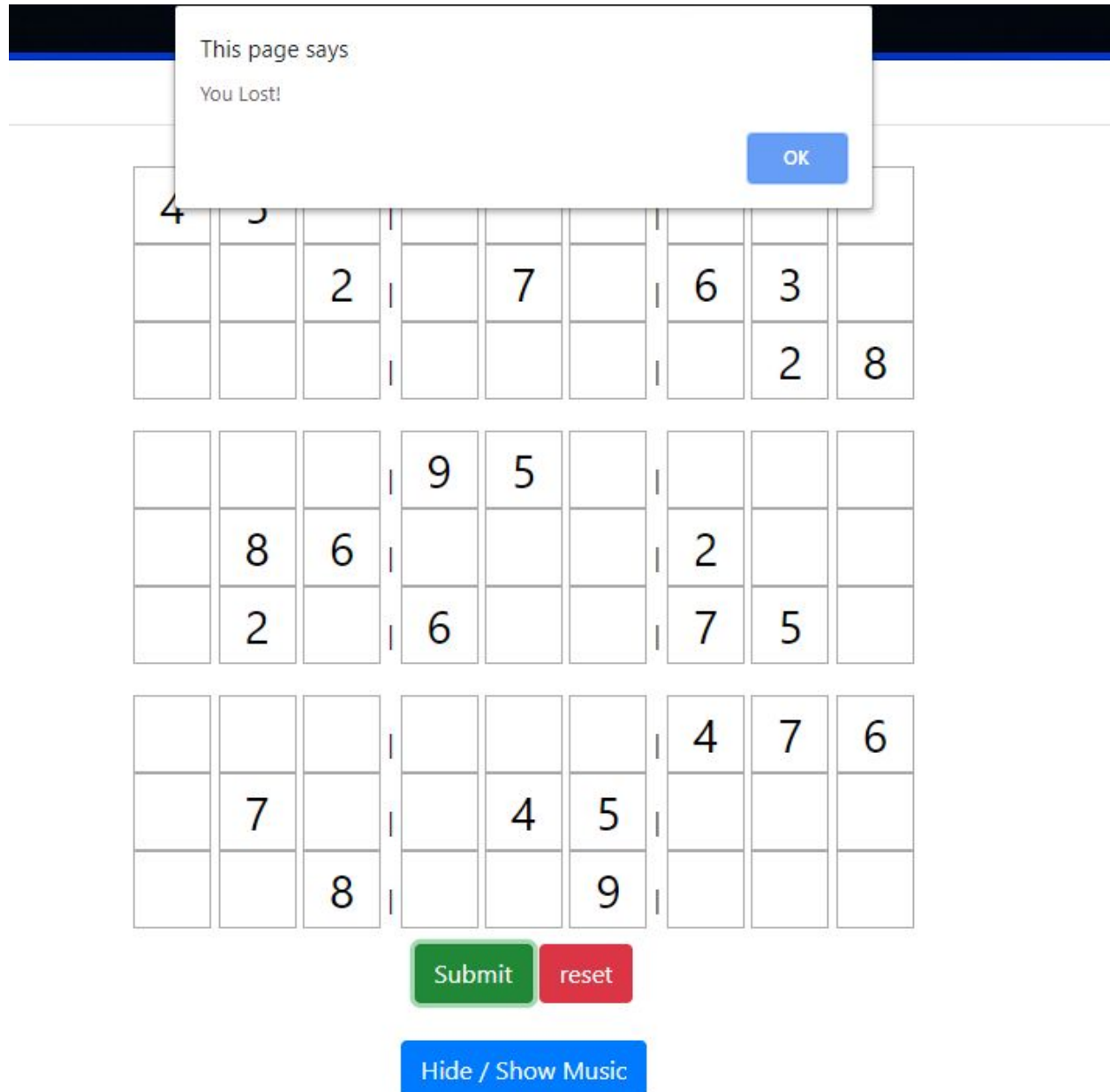


Figure 9- The interface showing that the user lost. It

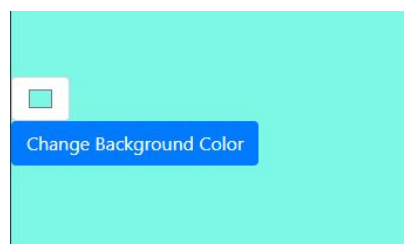
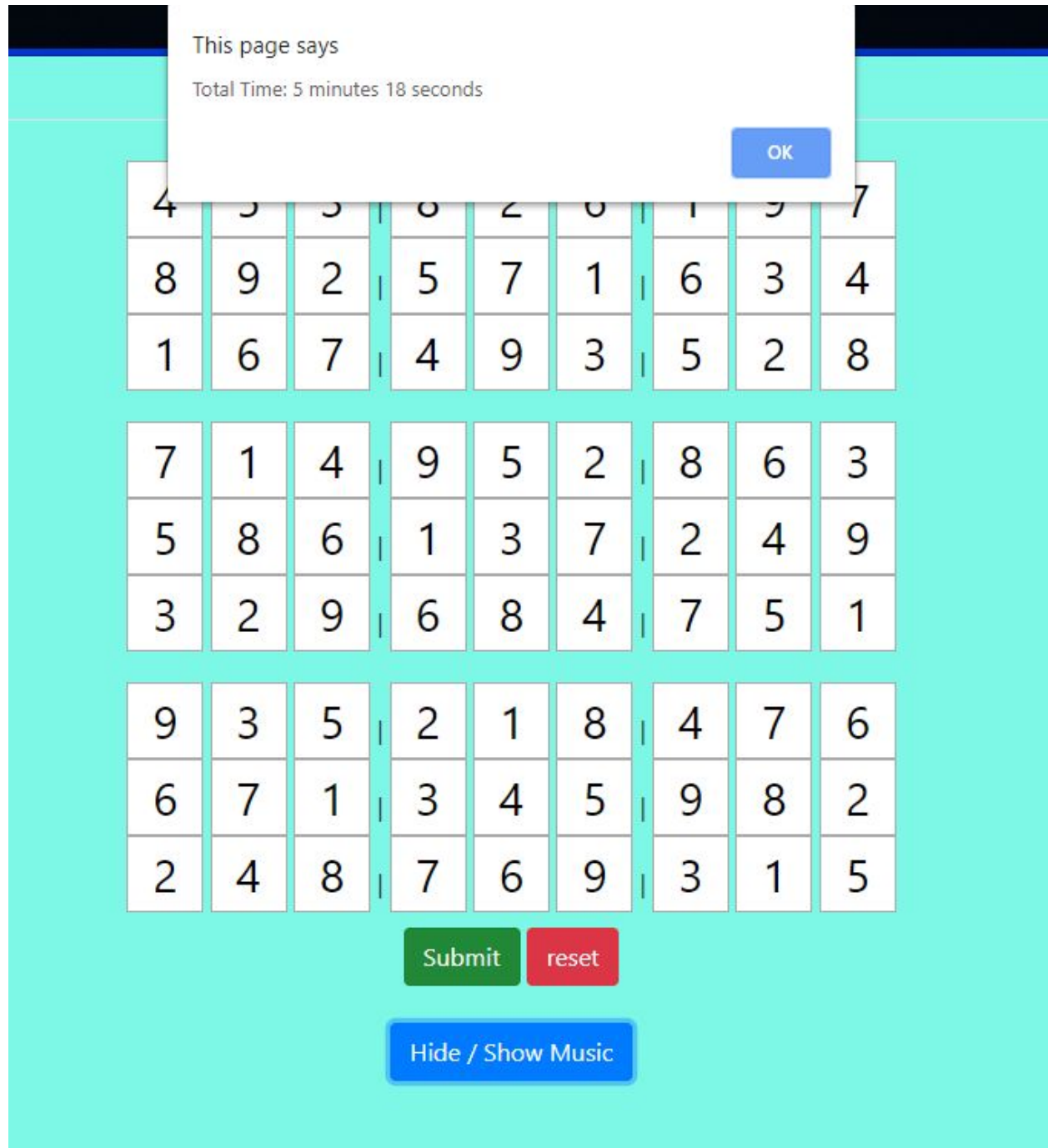


Figure 10- The interface with the change in background colour.

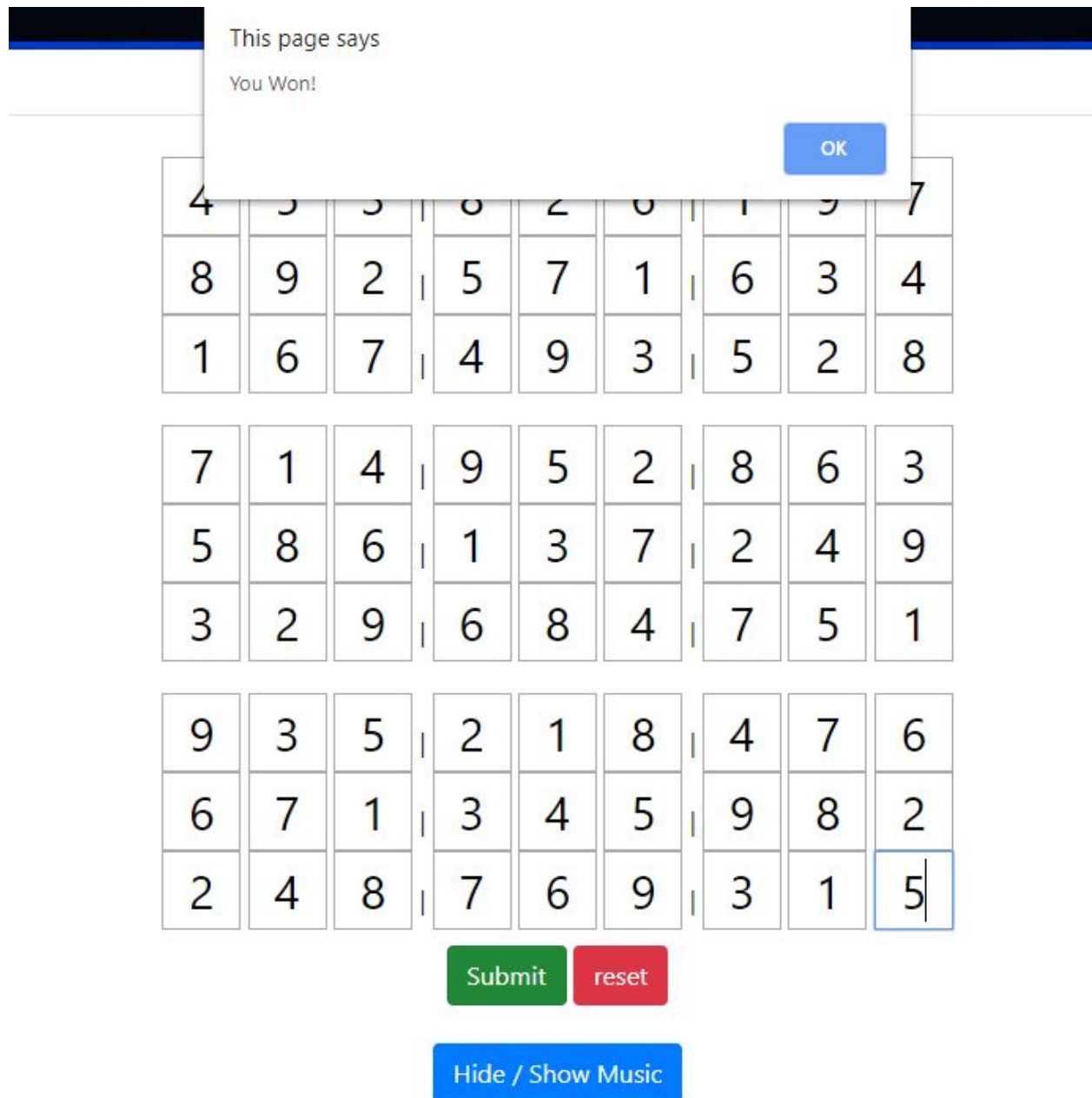


Figure 11- The display when the output is correct.



This page says

Please enter a username, it cannot be blank

OK

Username

Password

Email

Credit Card Number

submit

Figure 12- What the user views when the username is blank

This page says  
please Enter a Password

OK

Username

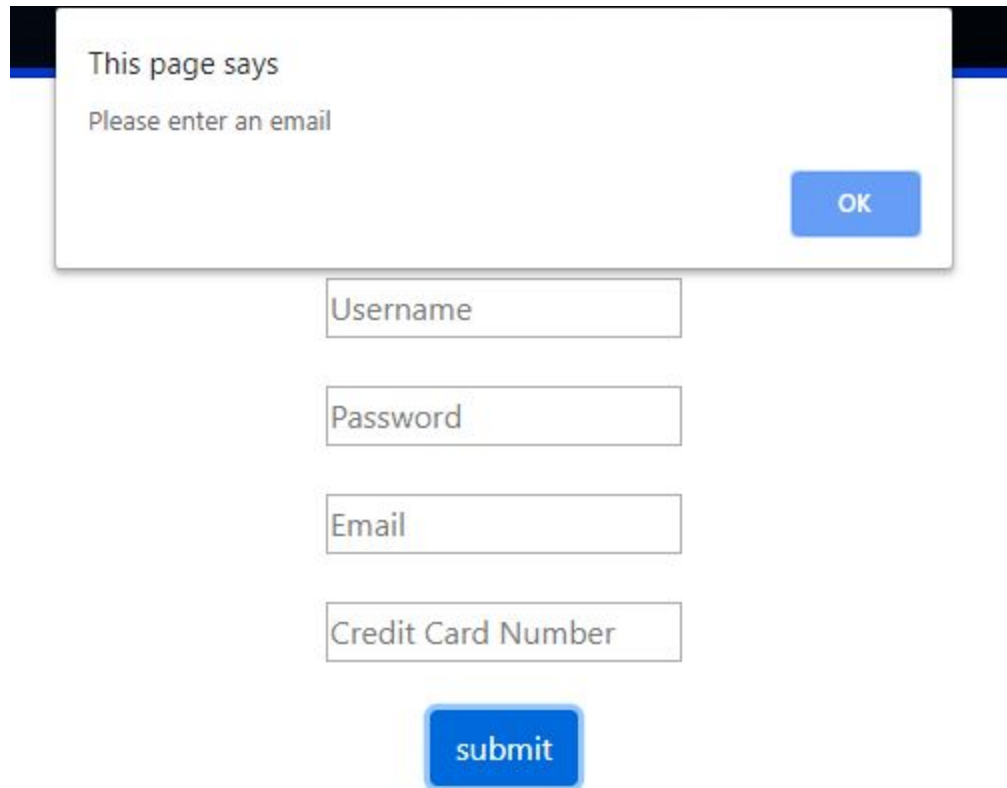
Password

Email

Credit Card Number

submit

Figure 13- What the user views when the password input is left blank.



This page says

Please enter an email

OK

Username

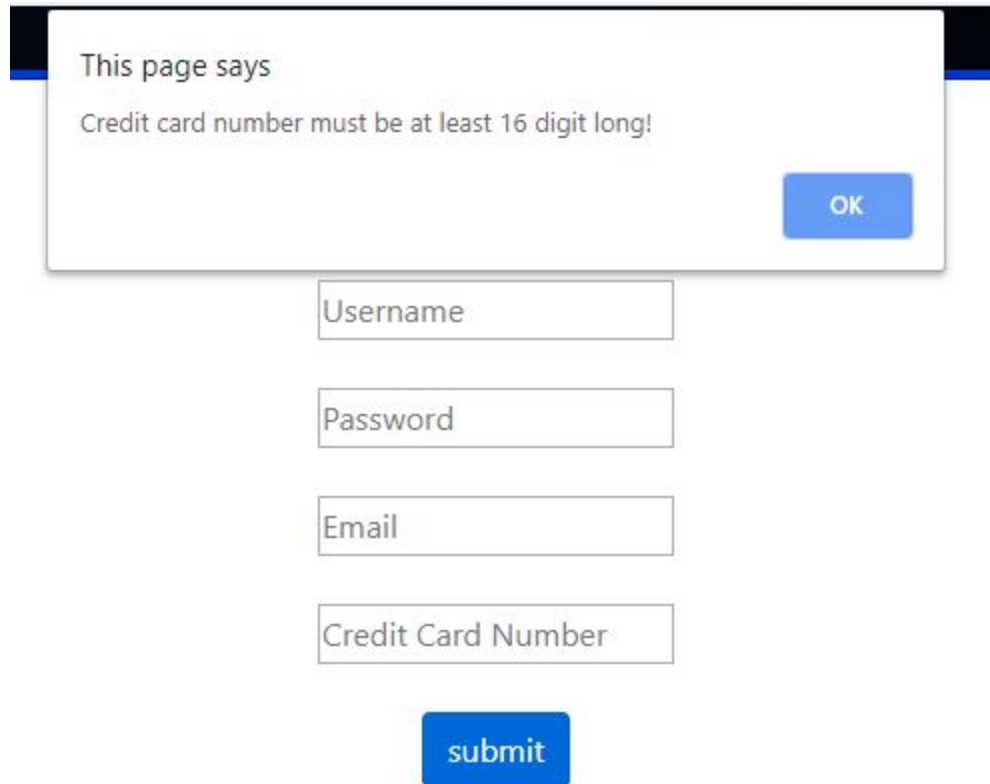
Password

Email

Credit Card Number

submit

Figure 14- Email is blank



This page says

Credit card number must be at least 16 digit long!

OK

Username

Password

Email

Credit Card Number

submit

Figure 15- Credit card number is blank

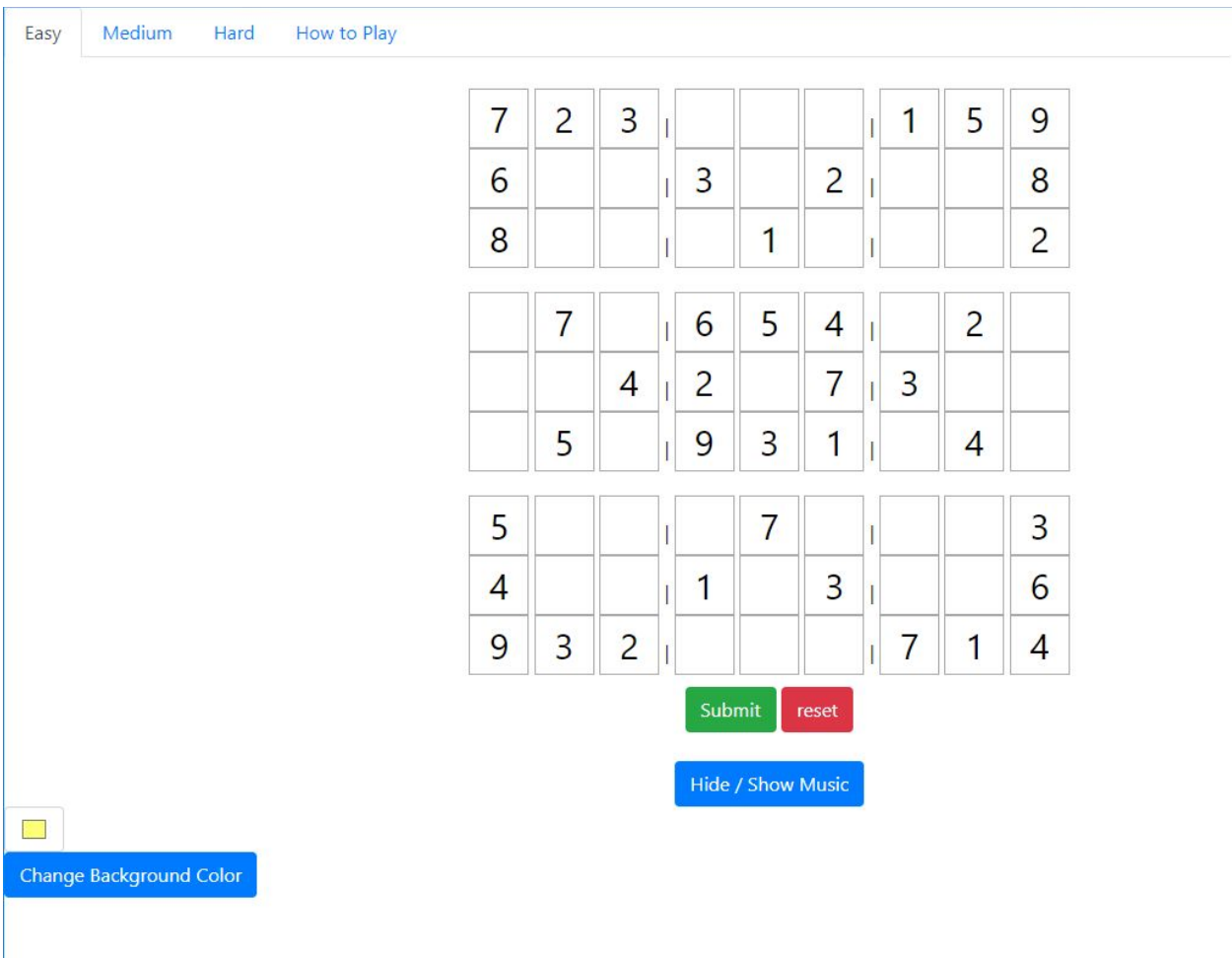


Figure 16- Easy, Medium, and hard difficulty levels available at the top

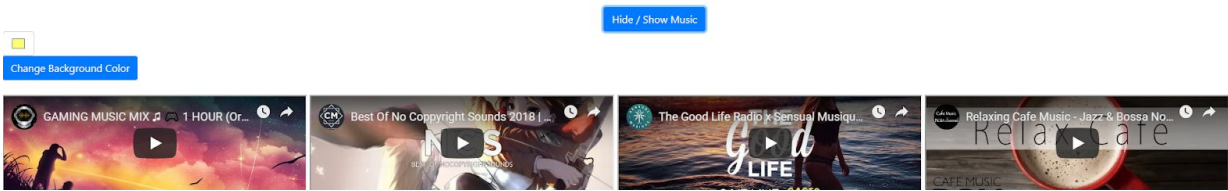


Figure 17- Before Hiding Music



Figure 17- After Hiding Music

7	2	3		4	4	1	5	9
6	6	6	3	5	2	1	1	8
8	3	4	5	1	8	5	5	2

6	7	9	6	5	4	5	2	8
7	8	4	2	1	7	3	1	1
5	5	7	9	3	1	6	4	6

5	6	6	6	7	6	5	5	3
4	5	5	1	5	3	5	5	6
9	3	2	5	5	5	7	1	4

Submit

reset

Hide / Show Music

Figure 18- Before Clicking Reset

---

7	2	3				1	5	9
6			3		2			8
8				1				2

	7		6	5	4		2	
		4	2		7	3		
	5		9	3	1		4	

5				7				3
4			1		3			6
9	3	2				7	1	4

Figure 19- After Clicking Reset

[Easy](#)
[Medium](#)
[Hard](#)
[How to Play](#)

7	2	3				1	5	9
6			3		2			8
8				1				2
	7		6	5	4		2	
		4	2		7	3		
	5		9	3	1		4	
5				7				3
4			1		3			6
9	3	2				7	1	4

Submit
reset

Hide / Show Music



Change Background Color

Figure 20- Easy Level



[Easy](#)
[Medium](#)
[Hard](#)
[How to Play](#)

6	5	9		1		2	8	
1				5			3	
2			8				1	
			1	3	5		7	
8			9					2
		3		7	8	6	4	
3		2			9			4
					1	8		
		8	7	6				

Submit
reset

Hide / Show Music

☐
Change Background Color

Figure 21- Medium Level

EasyMediumHardHow to Play

4	5							
		2		7		6	3	
							2	8

			9	5				
	8	6				2		
	2		6			7	5	

						4	7	6
	7			4	5			
		8			9			

Submitreset

Hide / Show Music

Change Background Color

Figure 22- Hard Level

Easy Medium Hard How to Play

## How to Play Sudoku

How to Play Sudoku for Absolute Beginners  
9 x 9 grid

Watch later Share

Horizontal Blocks

3 sets of horizontal blocks

Rank

2	4	9		7	6			
8	3			1			6	9
					9	5	2	
3	6		2	4			5	
	9			2		4	1	
	5	4	6				3	
	1	6	9			2		8
		7	1	3	8			5
9					5			1

Click to Start!

Figure 23- How to Play Video