Information and Networks Security

HW_01 Steganography

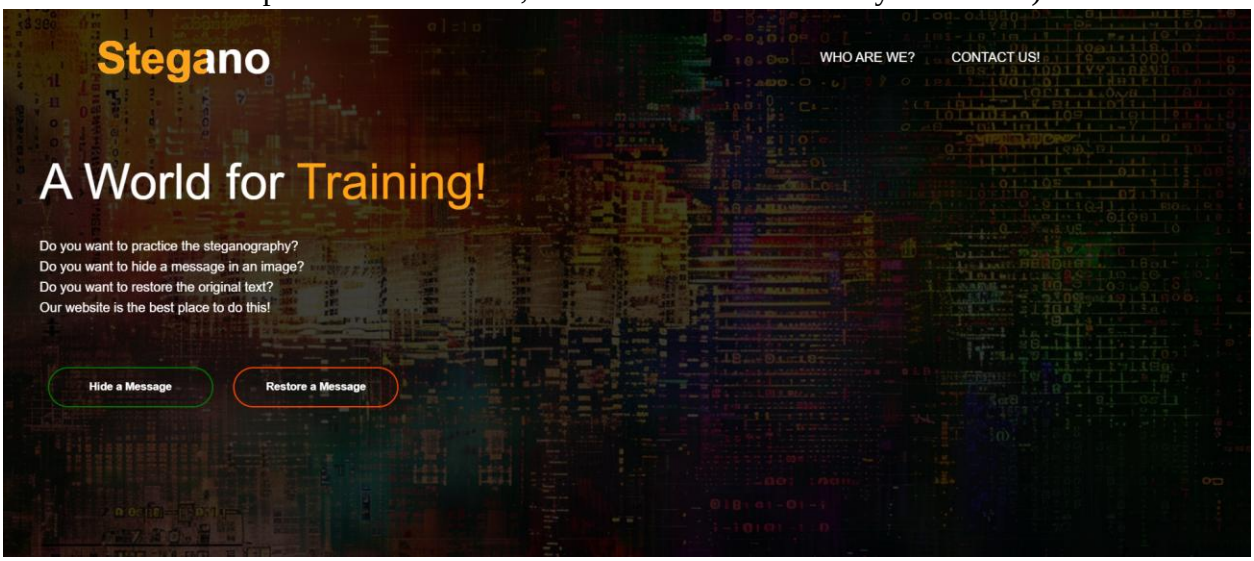
❖ Important Note

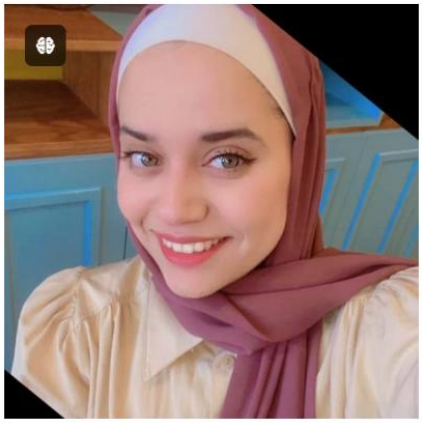
The URLs start with **http://127.0.0.1:5500** since i used Visual Studio Code Live Server in the developing of this website.

❖ Screenshots from the Website

➢ **Home Page**

Contains multiple animations such as (water-filling animation, typing animation, multiple labels animation, hover animations and many other ones).

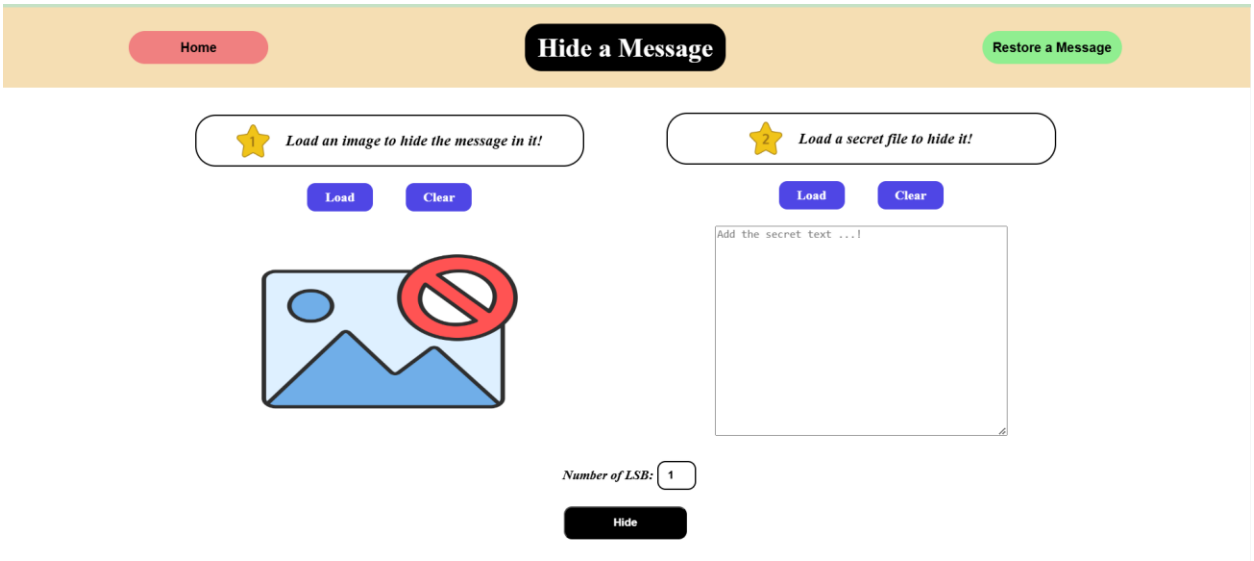


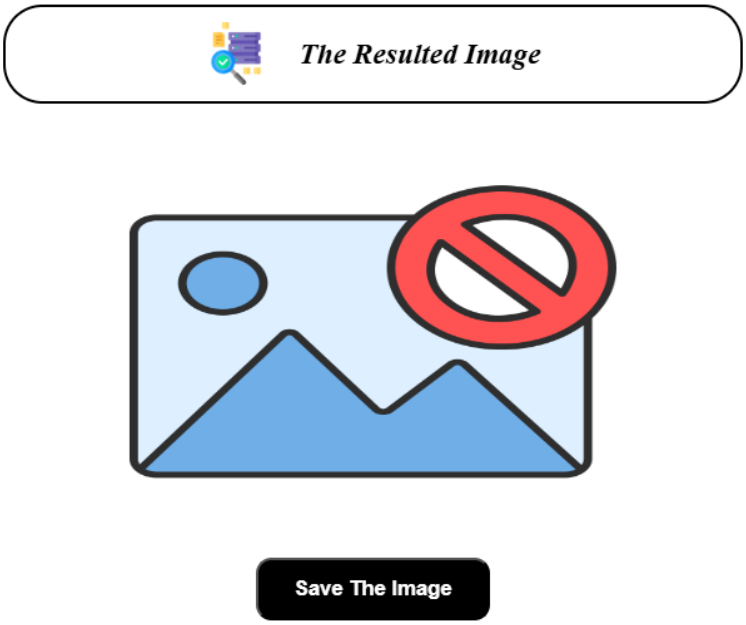


Najat Mansour

Computer Engineer

- **<u>Hide Page</u>**
  Contains the following:
  1. Header that contains a title and navigation buttons to the home page and restoring page.
  2. Section to upload an image contains two buttons to upload or clear.
  3. Section to upload a text-file contains two buttons to upload or clear.
  4. Text-field to determine the number of the number of LSBs used in hiding and restoring.
  5. Button to apply the hiding algorithm.
  6. Result section to display the modified image contains a button to save it on the local computer.





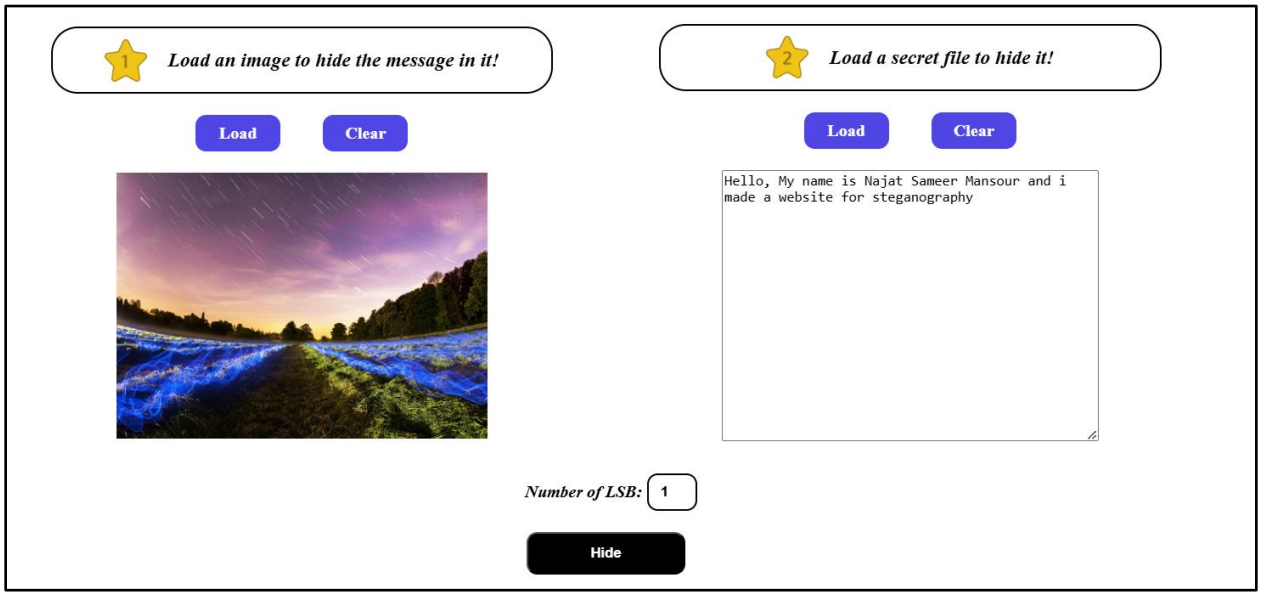
- **Restore Page**

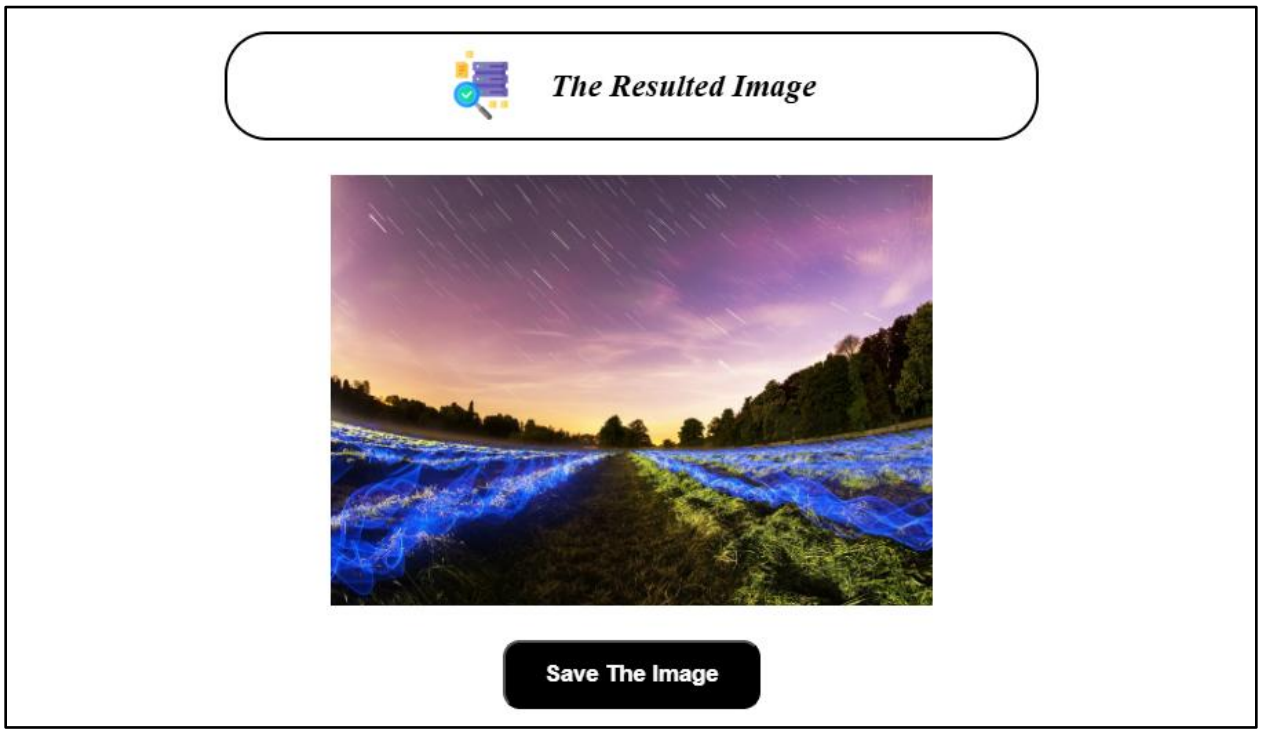
1. Contains the following:
2. Header that contains a title and navigation buttons to the home page and hiding page.
3. Section to upload an image contains two buttons to upload or clear.
4. Text-field to determine the number of the number of LSBs used in hiding and restoring.
5. Button to apply the restoring algorithm.
6. Result section to display the extracted secret text.

❖ Testcases using 1, 2 and 3 LSBs

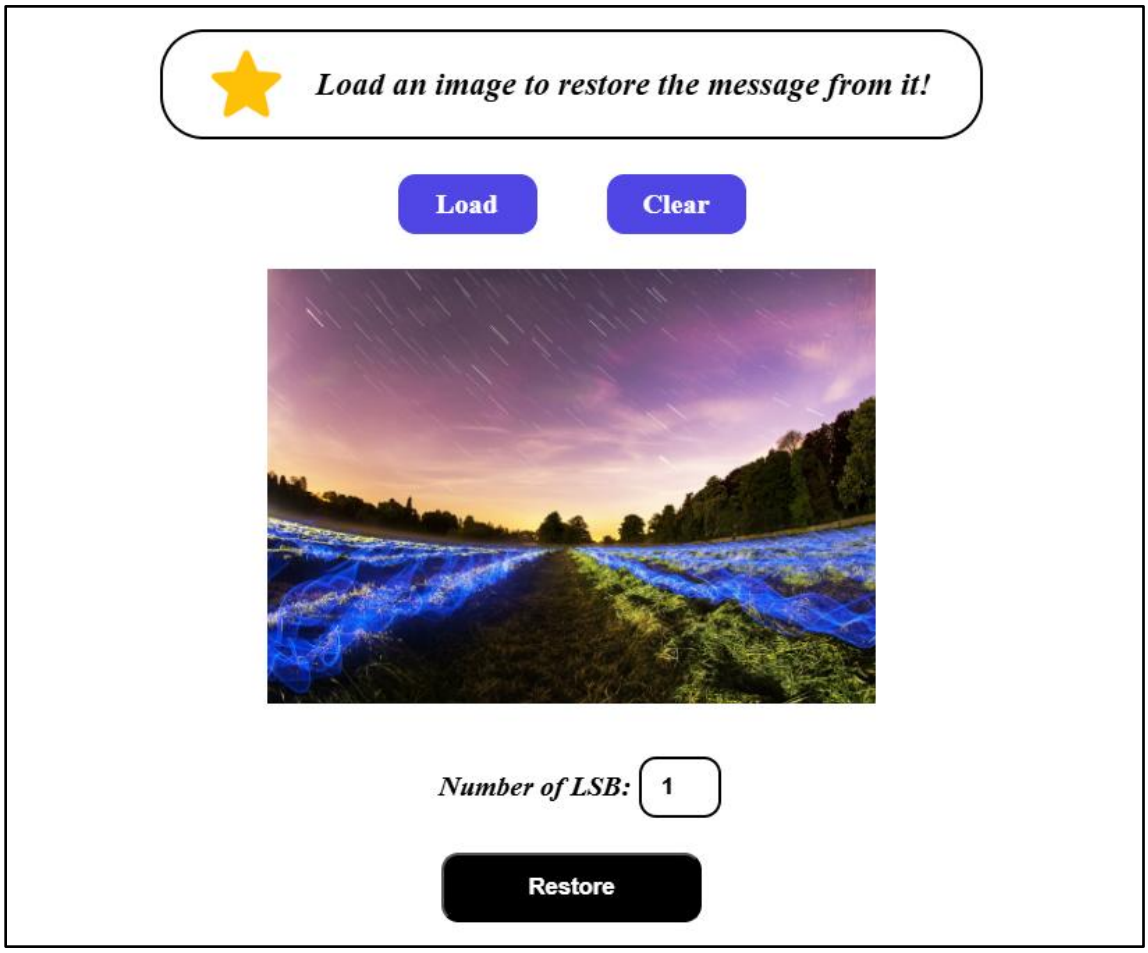
- **1 LSB**

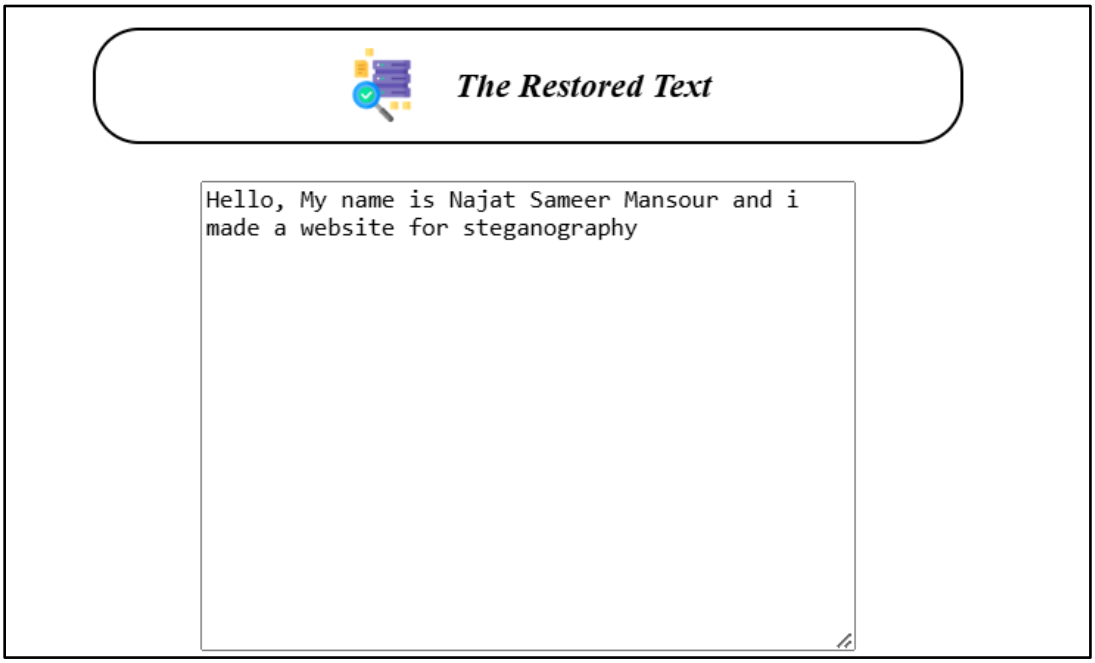
  First: Hide the secret



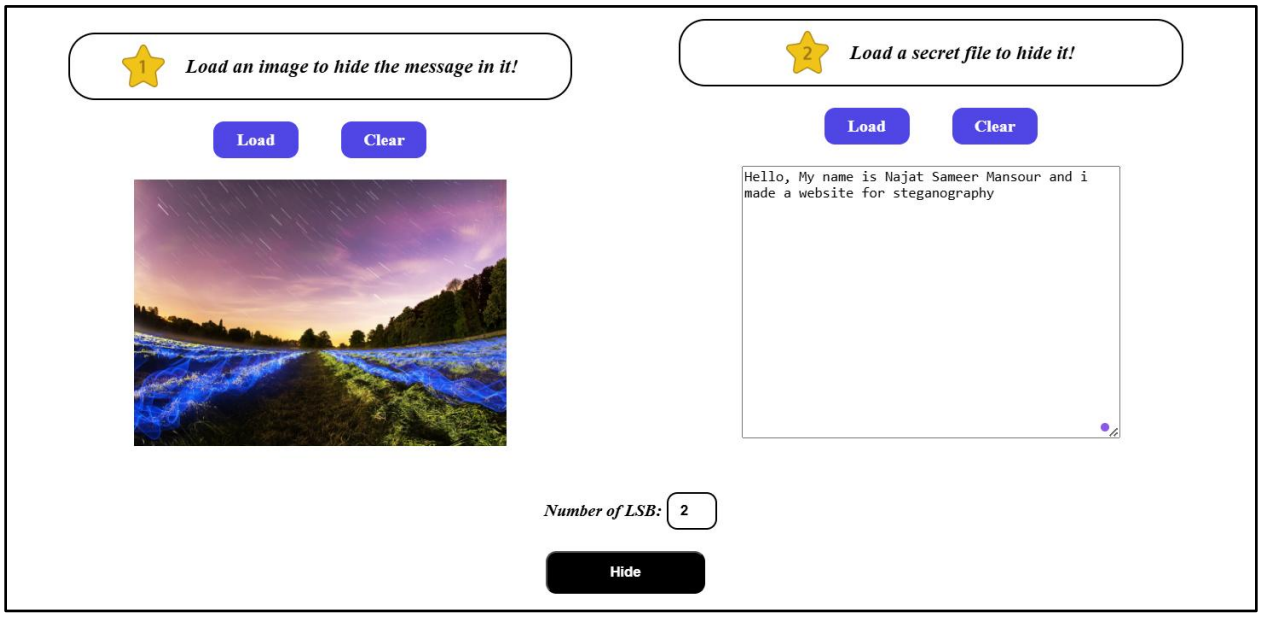


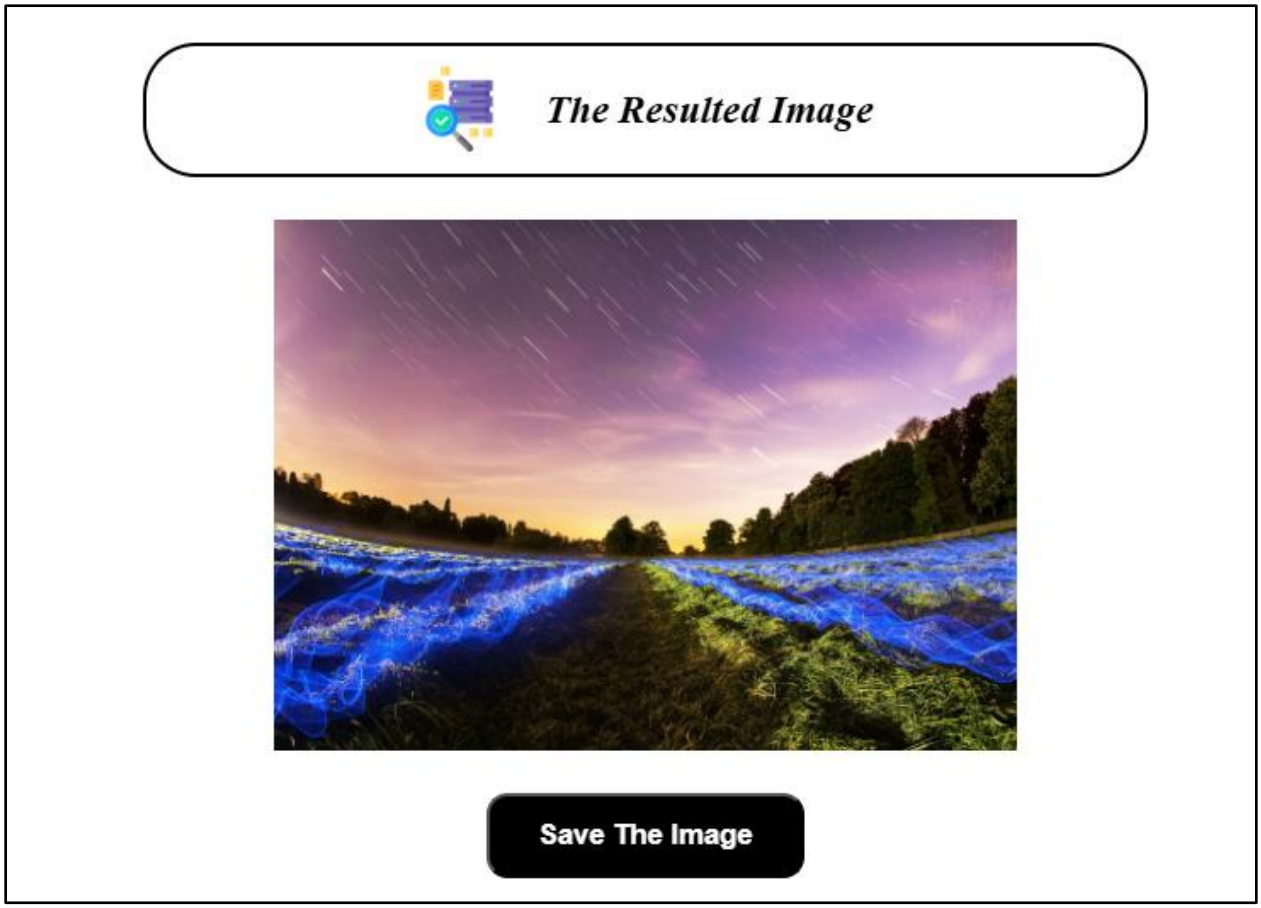
Second: Restore the secret



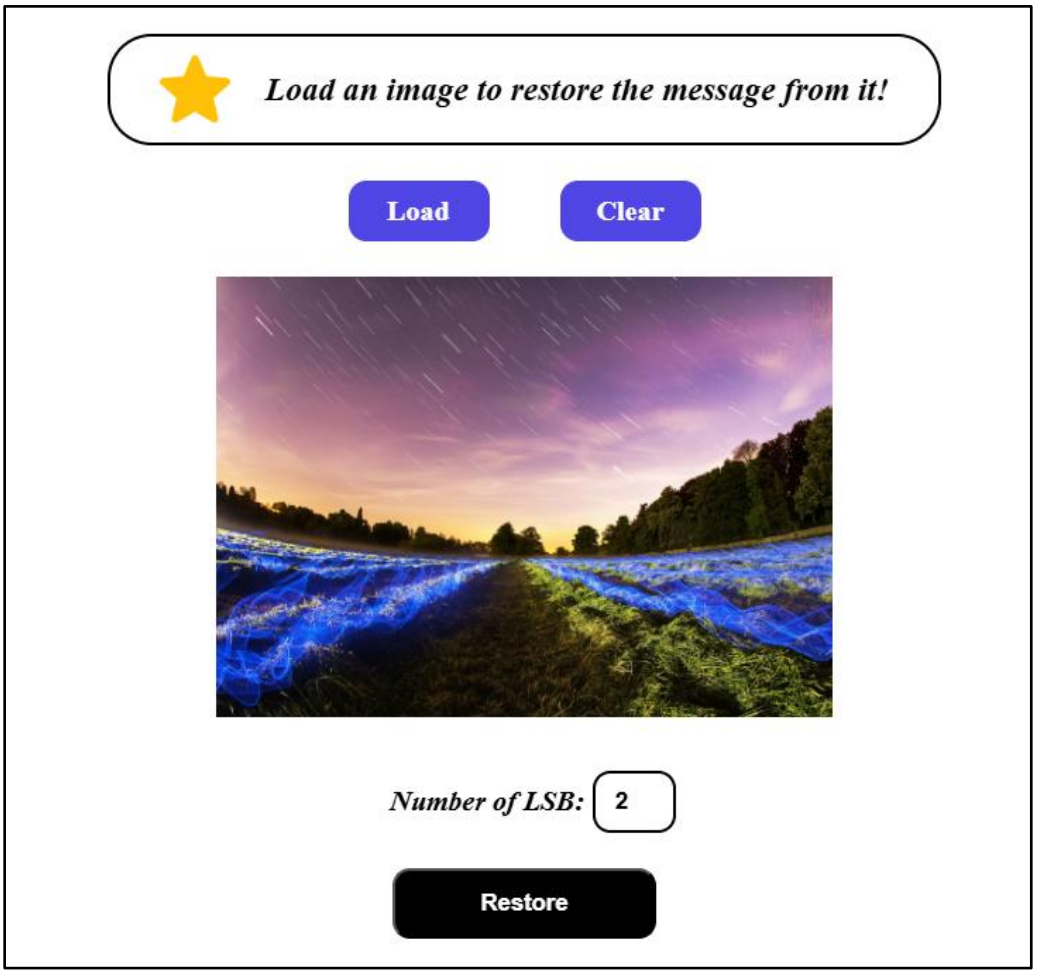


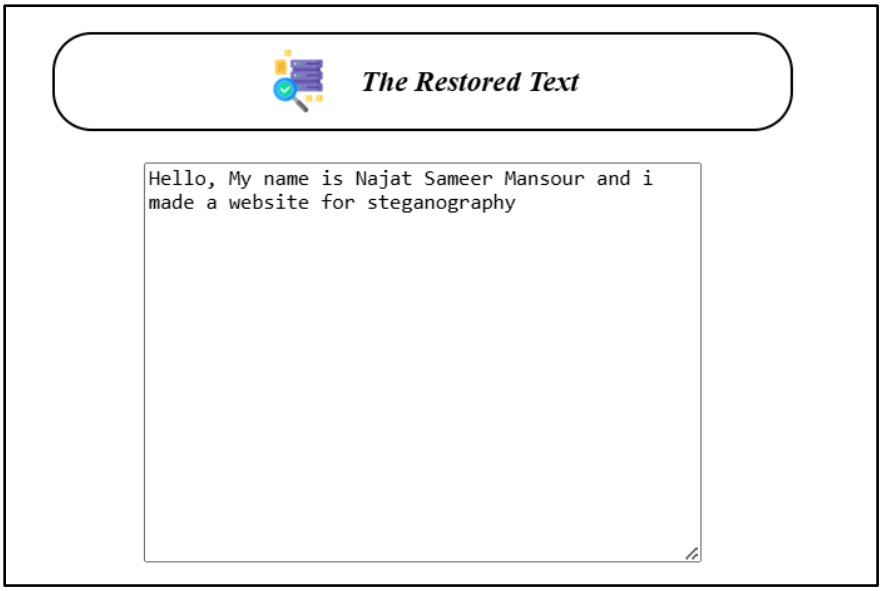
- **2 LSBs**
  First: Hide the secret





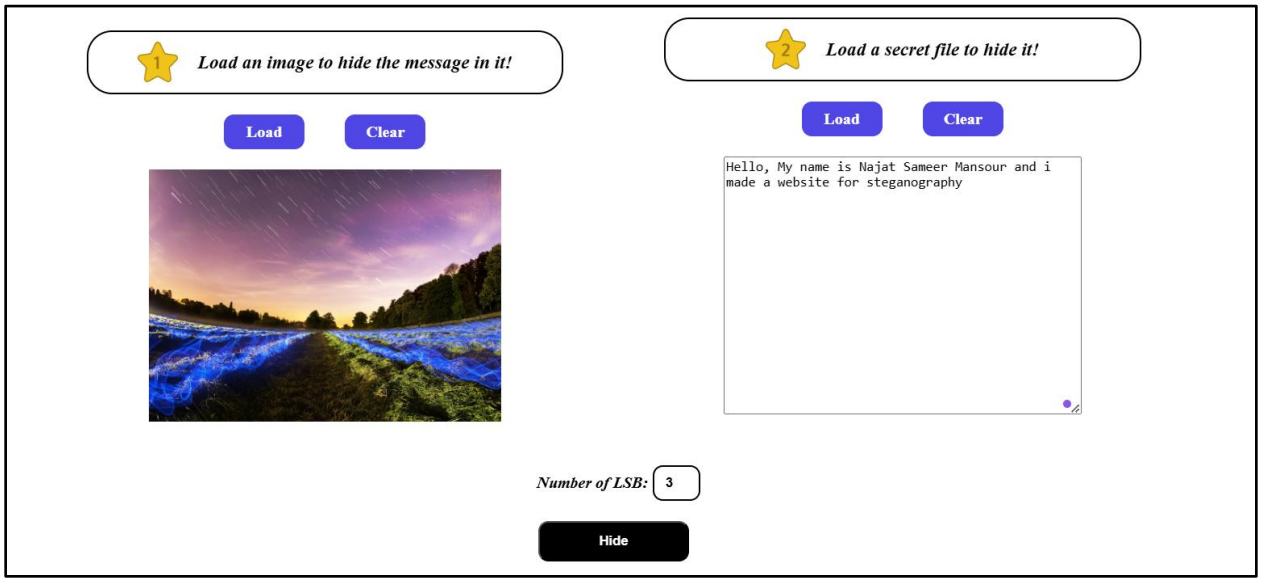
Second: Restore the secret

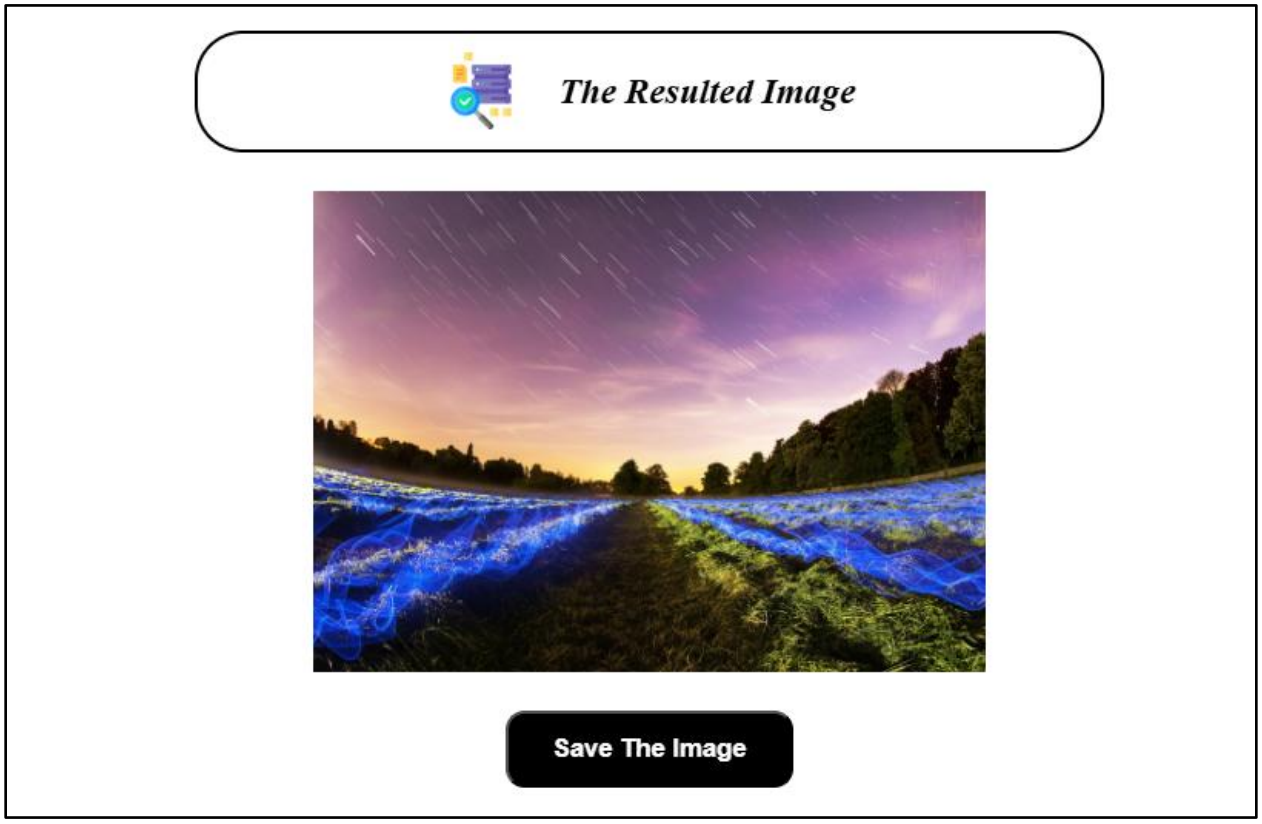




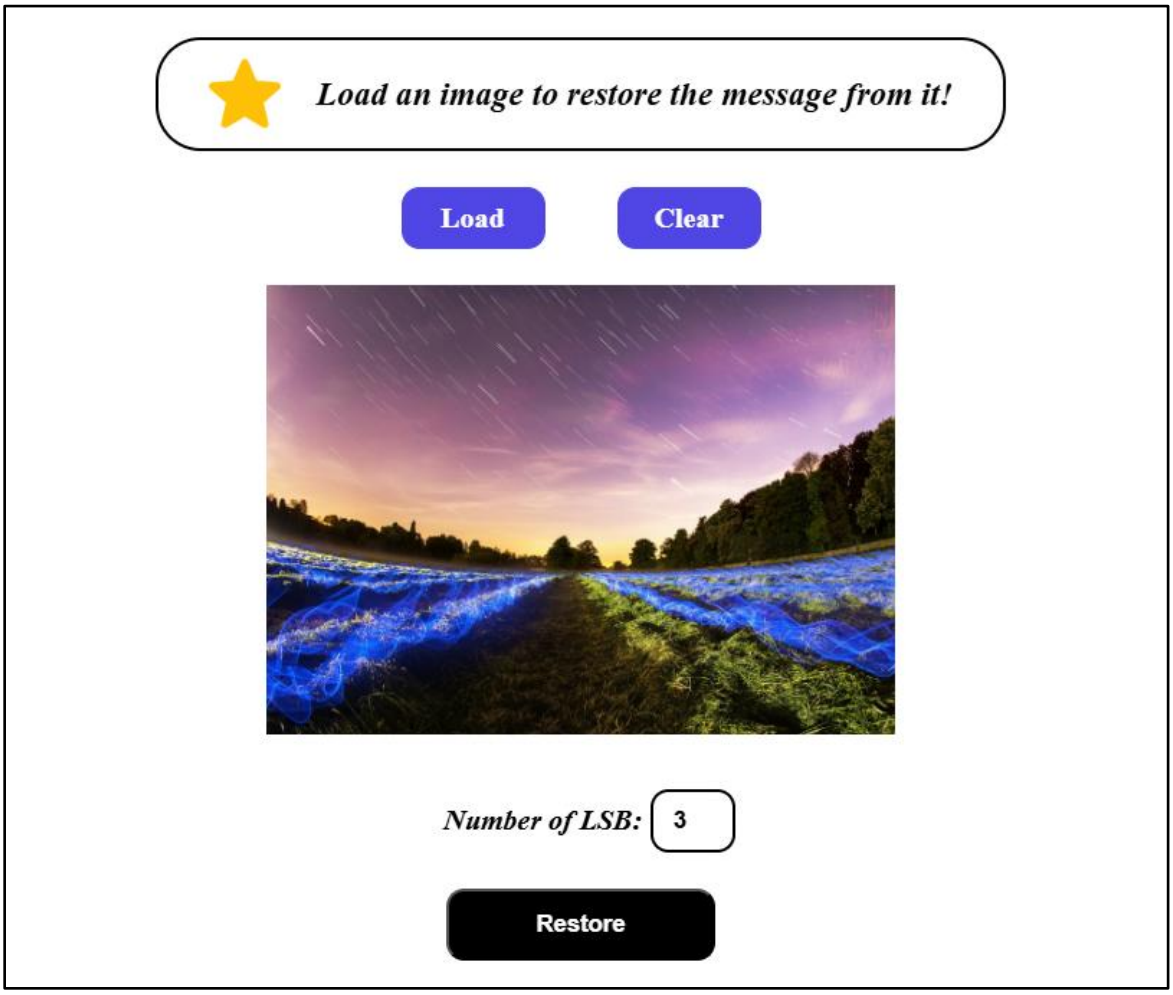
- **3 LSBs**

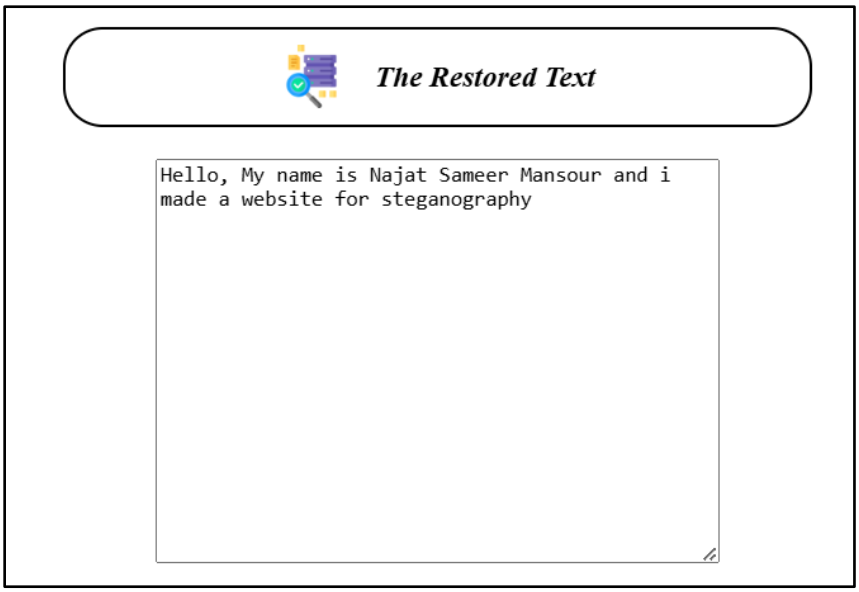
  First: Hide the secret





Second: Restore the secret





❖ Explaining Hiding and Restoring algorithms

➢ **Hiding Algorithm**

1. Check if all the input fields (Image, Secret text and the number of LSBs) are available or not.
2. Convert the secret text into array of bits.
3. Add **NULL** character (**0b00000000**) at the end of the secret's array of bits to indicate the end of it.
4. Convert the image into a buffer (array of bytes), make sure to skip the header of the BMP image which consists of 54 bytes.
5. Implementing the hiding algorithm as the following:
   A. Store the secret text's bits in the image LSBs using the bitwise OR after clearing them using the bitwise AND. I implemented three different cases to support using 1, 2 or 3 LSBs in the hiding and restoring instead of using the shifting to enhance the readability of the code.
   B. Create a BLOB (Binary Large Object) from the modified image's bytes and generate a URL for it to display the modified image in the result section of the website. I revoked the URL of the BLOB to cleanup the memory.

```
// Check if all the input fields (Image, Secret text and the number of LSBs) are available or not.
if (imgViewer.src === "http://127.0.0.1:5500/assets/imgs/no_image.png") {
    alert("Please, upload an image to be used!");
    return;
}

if (fileViewer.value === "") {
    alert("Please, enter a secret message to be hidden in the image!");
    return;
}

const numberOfLSBs = Number(txtLSBs.value);
if (numberOfLSBs < 1 || numberOfLSBs > 3) {
    alert("Please, enter a valid number of LSBs between 1 and 3!");
    return;
}
```

```
const secretMessage = fileViewer.value;

// Convert the secret text into array of bit.
const messageBits = [];
for (let i = 0; i < secretMessage.length; i++) {
    const charCode = secretMessage.charCodeAt(i);
    for (let j = 7; j >= 0; j--) {
        messageBits.push((charCode >> j) & 1);
    }
}

// Add NULL character (0b00000000) at the end of the secret's array of bits to indicate the end of it.
for (let i = 0; i < 8; i++) {
    messageBits.push(0);
}
```

```javascript
// Convert the image into a buffer (array of bytes)
fetch(imgViewer.src)
    .then(response => response.arrayBuffer())
    .then(buffer => {
        const imageData = new Uint8Array(buffer);
        // Determining where the actual bytes of the image starts from (Leave the header's 54 bytes)
        const imagePixelsStartsFrom =
            imageData[10]              // First Byte
            + (imageData[11] << 8)     // Second Byte
            + (imageData[12] << 16)    // Third Byte
            + (imageData[13] << 24);   // Fourth Byte

        // Hiding Process
        let messageBitIndex = 0;

        // A) Store the secret text's bits in the image LSBs using the bitwise OR after clearing them using the bitwise AND.
        // I implemented three different cases to support using 1, 2 or 3 LSBs in the hiding and restoring.
        for (let i = imagePixelsStartsFrom; i < imageData.length && messageBitIndex < messageBits.length; i++) {
            if (numberOfLSBs === 1) {
                imageData[i] &= 0b11111110;
                imageData[i] |= messageBits[messageBitIndex++];

            } else if (numberOfLSBs === 2) {
                imageData[i] &= 0b11111100;
                imageData[i] |= (messageBits[messageBitIndex++] << 1) | (messageBits[messageBitIndex++]);

            } else if (numberOfLSBs === 3) {
                imageData[i] &= 0b11111000;
                imageData[i] |= (messageBits[messageBitIndex++] << 2) | (messageBits[messageBitIndex++] << 1) | (messageBits[messageBitIndex++]);
            }
        }
```

```javascript
            // B) Create a BLOB (Binary Large Object) and generate a URL for it
            const blob = new Blob([imageData], { type: "image/bmp" });
            const newImageUrl = URL.createObjectURL(blob);

            // Memory cleanup optimization
            if (resultedImage.src) {
                URL.revokeObjectURL(resultedImage.src);
            }

            // Display the image
            resultedImage.src = newImageUrl;
        })
        .catch(error => console.error("Error loading the image:", error));
});
```

- **<u>Restoring Algorithm</u>**

  1. Check if all the input fields (Image and the number of LSBs) are available or not.
  2. Convert the image into a buffer (array of bytes), make sure to skip the header of the BMP image which consists of 54 bytes.
  3. Implement the restoring algorithm as the following:
     A. Getting the bits stored in the LSBs of the image using the bitwise AND as well as shifting (It's better here to use the shifting instead of writing three different cases as I did in the hiding part).

B. Convert the bits into bytes (8-bits) and generate ASCII characters from them.
C. The stop condition is getting a **NULL** character (**0b00000000**).
D. Display the generated secret text on the text area to allow the user to see it.

```
// Check if all the input fields (Image and the number of LSBs) are available or not.
if (imgViewer.src === "http://127.0.0.1:5500/assets/imgs/no_image.png") {
    alert("Please, upload an image to be used!");
    return;
}

if (txtLSBs.value === "") {
    alert("Please, fill the number of used LSBs!");
    return;
}

const numberOfLSBs = Number(txtLSBs.value);
if (numberOfLSBs < 1 || numberOfLSBs > 3) {
    alert("Please, enter a valid number of LSBs between 1 and 3!");
    return;
}
```

```
    // Convert the image into a buffer (array of bytes)
    fetch(imgViewer.src)
        .then(response => response.arrayBuffer())
        .then(buffer => {
            const imageData = new Uint8Array(buffer);
            // Determining where the actual bytes of the image starts from (Leave the header which is 54-bytes)
            const imagePixelsStartsFrom =
                imageData[10]              // First Byte
                + (imageData[11] << 8)     // Second Byte
                + (imageData[12] << 16)    // Third Byte
                + (imageData[13] << 24);   // Fourth Byte

            let messageBits = [];
            let resultText = "";

            // Restoring Process
            for (let i = imagePixelsStartsFrom; i < imageData.length; i++) {
                // A) Getting the bits stored in the LSBs of the image using the bitwise AND as well as shifting
                for (let j = numberOfLSBs - 1; j >= 0; j--) {
                    messageBits.push((imageData[i] >> j) & 1);
                }

                // B) Convert the bits into bytes (8-bits) and generate ASCII characters from them.
                if (messageBits.length >= 8) {
                    let charCode = 0;
                    for (let b = 0; b < 8; b++) {
                        charCode = (charCode << 1) | messageBits[b];
                    }

                    // Check for NULL char ==> It terminates the secret message
                    if (charCode === 0) break;
```

```
                resultText += String.fromCharCode(charCode);
                messageBits = messageBits.slice(8);
            }
        }

        // Show the secret message on the text area
        document.querySelector("textarea").value = resultText;
    })
    .catch(error => console.error("Error loading the image:", error));
});
```

THE END!