**eMAIL ADDRESS VALIDATOR ACTIVITY**

**Minor Project**

*for*

**ELEWAYTE INTERNSHIP**

*By*

Maru Hasini Reddy   
Email: [hasinireddy286@gmail.com](mailto:hasinireddy286@gmail.com)  
  
  
  
  
  
  
  
  
  
  
  
**Investigation (Rules for Email Validation):**1. An email address must contains an @ symbol, which separates the local part from the domain part of the address.

2. The @ symbol cannot be consecutive

3. There cannot be more than one @ symbol in an email

4. The @ symbol is allowed in between single quotes but if another one is added in the email it won’t be valid

5. The local part of the email address (before the @ symbol) can contain letters, numbers, and special characters such as !#$%&'\*+-/=?^\_{|}~, as well as periods (.`), as long as they are not the first or last character and do not appear consecutively.

6. The domain part of the email address (after the @ symbol) must be a valid domain name, which means it must contain at least one period (.) and consist of letters, numbers, and hyphens (-).

7. The top-level domain (the part after the last period) must be a valid top-level domain such as .com, .org, or .edu.

8. The total length of an email address must not exceed 254 characters.

9. The @ symbol has to be more than 6 letters from the end of the email

10. @ symbol is a must

11. No spaces between

12. The local name may be a maximum of 64 letters long

13. Icon Characters aren’t allowed

14. Underscore is not allowed in domain part

15. Spaces, quotes and backslashes must be contained by quotes

16. Last portion of domain must be at least two characters e.g. .com

17. Punctuation marks cannot come consecutively

18. Can’t have all punctuation marks

19. Must have the full stop separating the local and domain name

20. @ sign can’t be first

21. Hyphens can’t be first or last character

22. Normally capital and lowercase letter versions of email work and are allowed

23. Space between “ ” is allowed

24. Any punctuation can go in quotes

25. IPV4 addresses to be checked Eg: 123.123.123.123

[197.0.27.255] are valid IPV4 domain addresses  
  
For satisfying above rules, I written a function using RegEx in python.

import re

def is\_valid\_email(email):

    email\_regex = r"""(?:[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])\*")@(?:(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?|\[(?:(?:(2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-9]))\.){3}(?:(2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-9])|[a-z0-9-]\*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])"""

    res = re.match(email\_regex, email)

    return res is not None

**Explanation:**   
**import re**: This line imports the **re** module in Python, which provides support for regular expressions.

1. **def is\_valid\_email(email):**: This line defines a function named **is\_valid\_email** that takes an **email** parameter, representing the email address to be validated.
2. **email\_regex**: This variable holds a long regular expression pattern that defines the structure and format of a valid email address. The regular expression pattern is stored as a raw string (denoted by the **r** prefix), allowing for easier handling of backslashes and special characters.
3. **Regular Expression Pattern**: The regular expression pattern is a complex sequence of characters that defines the structure of a valid email address. It matches different components of an email address, including the local part, the "@" symbol, and the domain part. The pattern is quite lengthy and intricate due to the complexity of email address validation.

To summarize, the regular expression pattern checks for the following components of a valid email address:

* + The local part (username) before the "@" symbol.
  + The "@" symbol.
  + The domain part (including the top-level domain).

1. **res = re.match(email\_regex, email)**: This line uses the **re.match** function to attempt to match the provided **email** against the defined regular expression pattern (**email\_regex**). If the **email** matches the pattern, a match object is returned; otherwise, **None** is returned.
2. **return res is not None**: The function returns **True** if the regular expression matches the provided **email**, indicating that the email is valid. It returns **False** if the regular expression does not match, indicating that the email is not valid.

In summary, the **is\_valid\_email** function uses a regular expression to validate whether the input string (**email**) adheres to the structure and format of a valid email address.

**app.py**

from flask import Flask, render\_template, request, jsonify

from validate\_email import is\_valid\_email

app = Flask(\_\_name\_\_)

@app.route('/', methods=['GET', 'POST'])

def email\_validator():

    if request.method == 'POST':

        email = request.form['email']

        is\_valid = is\_valid\_email(email)

        return jsonify({'is\_valid': is\_valid})

    return render\_template('email\_validator.html')

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

The provided code is a Flask application that uses the **is\_valid\_email** function to validate an email address sent via a POST request and returns the validation result as JSON. Let's break down how the **is\_valid\_email** function is used in this Flask application:

1. **Importing Necessary Modules**:

from flask import Flask, render\_template, request, jsonify from validate\_email import is\_valid\_email

* + The Flask module is imported to create a Flask application.
  + The **render\_template**, **request**, and **jsonify** functions are imported from Flask to handle rendering HTML templates, processing HTTP requests, and formatting JSON responses, respectively.
  + The **is\_valid\_email** function from the **validate\_email** module (presumably containing the **is\_valid\_email** function implementation) is imported to perform email validation.

1. **Creating a Flask Application**:

app = Flask(\_\_name\_\_)

* + A Flask application instance named **app** is created.

1. **Defining a Route for Handling Requests**:

@app.route('/', methods=['GET', 'POST']) def email\_validator(): ...

* + A route is defined for the root URL ("/"). This route handles both GET and POST requests.

1. **Request Handling and Email Validation**:

if request.method == 'POST': email = request.form['email'] is\_valid = is\_valid\_email(email) return jsonify({'is\_valid': is\_valid})

* + If the request method is POST, the email address is extracted from the form data sent in the request.
  + The **is\_valid\_email** function is called with the extracted email address to determine its validity.
  + The result of the email validation (**is\_valid**) is then returned as a JSON response.

1. **Rendering a Template**:

return render\_template('email\_validator.html')

* + If the request method is GET, the application renders an HTML template named **'email\_validator.html'**.

1. **Running the Flask Application**:

if \_\_name\_\_ == '\_\_main\_\_': app.run(debug=True)

* + If the script is run directly (as opposed to being imported), the Flask application is started with debugging enabled.

In summary, the Flask application handles GET and POST requests to the root URL. For POST requests, it validates the provided email address using the **is\_valid\_email** function and returns the validation result as JSON. For GET requests, it renders an HTML template. The **is\_valid\_email** function, previously defined, is used to determine the validity of the email address provided in the POST request.  
  
**email\_validator.html**

<!DOCTYPE html>

<html>

<head>

  <title>Email Validator</title>

  <style>

    body {

    font-family: Arial, sans-serif;

  }

  .container {

    width: 50%;

    margin: auto;

    padding: 20px;

  }

  input[type="text"] {

    width: 70%;

    padding: 10px;

    margin: 5px 0 20px 0;

    display: inline-block;

    border: 1px solid #ccc;

  }

  button {

    width: 30%;

    padding: 10px;

    display: inline-block;

    border: none;

    background-color: #4CAF50;

    color: white;

    cursor: pointer;

  }

  button:hover {

    opacity: 0.8;

  }

  #validationResult {

    margin-top: 20px;

    font-weight: bold;

  }

  </style>

</head>

<body>

  <div class="container">

    <h1>Email Validator</h1>

    <form id="emailForm" action="/" method="post">

      <label for="email">Enter an email address:</label><br>

      <input type="text" id="email" name="email" placeholder="Enter email">

      <button type="submit">Validate</button>

    </form>

    <div id="validationResult"></div>

  </div>

  <script>

    document.getElementById('emailForm').addEventListener('submit', function(event) {

      event.preventDefault();

      const email = document.getElementById('email').value;

      fetch('/', {

        method: 'POST',

        headers: {

          'Content-Type': 'application/x-www-form-urlencoded'

        },

        body: new URLSearchParams({email: email})

      })

      .then(response => response.json())

      .then(data => {

        const validationResult = data.is\_valid ? 'Valid email' : 'Invalid email';

        document.getElementById('validationResult').innerText = validationResult;

      });

    });

  </script>

</body>

</html>

This HTML document creates a simple web page that allows users to enter an email address and validate it using the provided Flask application. Here's a breakdown of the HTML document:

1. **DOCTYPE and HTML Tag**:

<!DOCTYPE html> <html>

* + The **DOCTYPE** declaration specifies the HTML version being used (HTML5 in this case).
  + The **<html>** tag marks the start of the HTML document.

1. **Head Section**:

<head> <title>Email Validator</title> <style> ... (CSS styles) </style> </head>

* + The **<head>** section contains metadata and styles for the HTML document.
  + The **<title>** tag sets the title of the web page to "Email Validator."
  + The **<style>** block contains CSS styles for formatting the elements in the document.

1. **Body Section**:

<body> ... (content) </body> </html>

* + The **<body>** section contains the main content of the web page.

1. **CSS Styles**:

body { font-family: Arial, sans-serif; } ...

* + Various CSS styles are defined to format the appearance of the elements in the document, including font styles, input fields, buttons, and validation results.

1. **Container and Form**:

<div class="container"> <h1>Email Validator</h1> <form id="emailForm" action="/" method="post"> ... (form content) </form> <div id="validationResult"></div> </div>

* + A **<div>** element with the class "container" contains the main content.
  + The **<h1>** tag displays the heading "Email Validator."
  + A **<form>** element with the ID "emailForm" is used to capture user input.

1. **Form Content (Input and Button)**:

<label for="email">Enter an email address:</label><br> <input type="text" id="email" name="email" placeholder="Enter email"> <button type="submit">Validate</button>

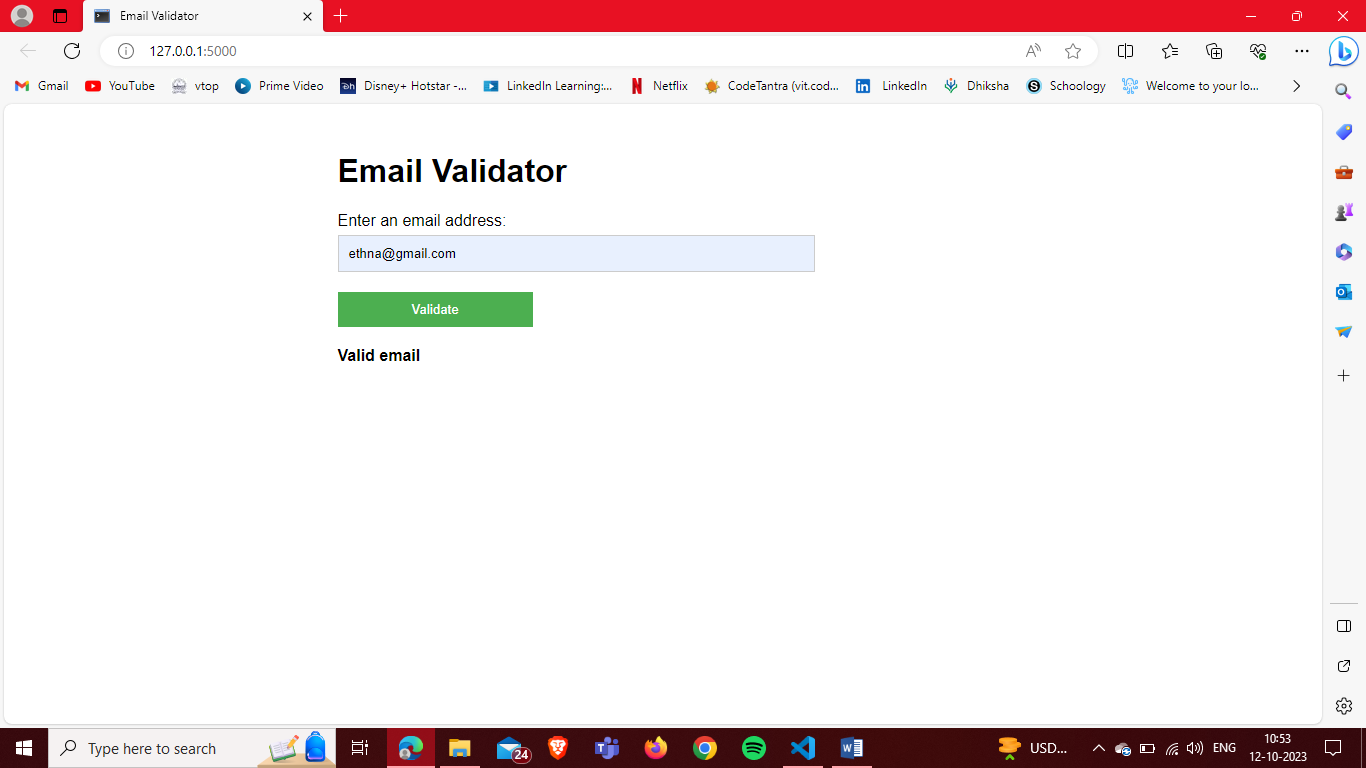
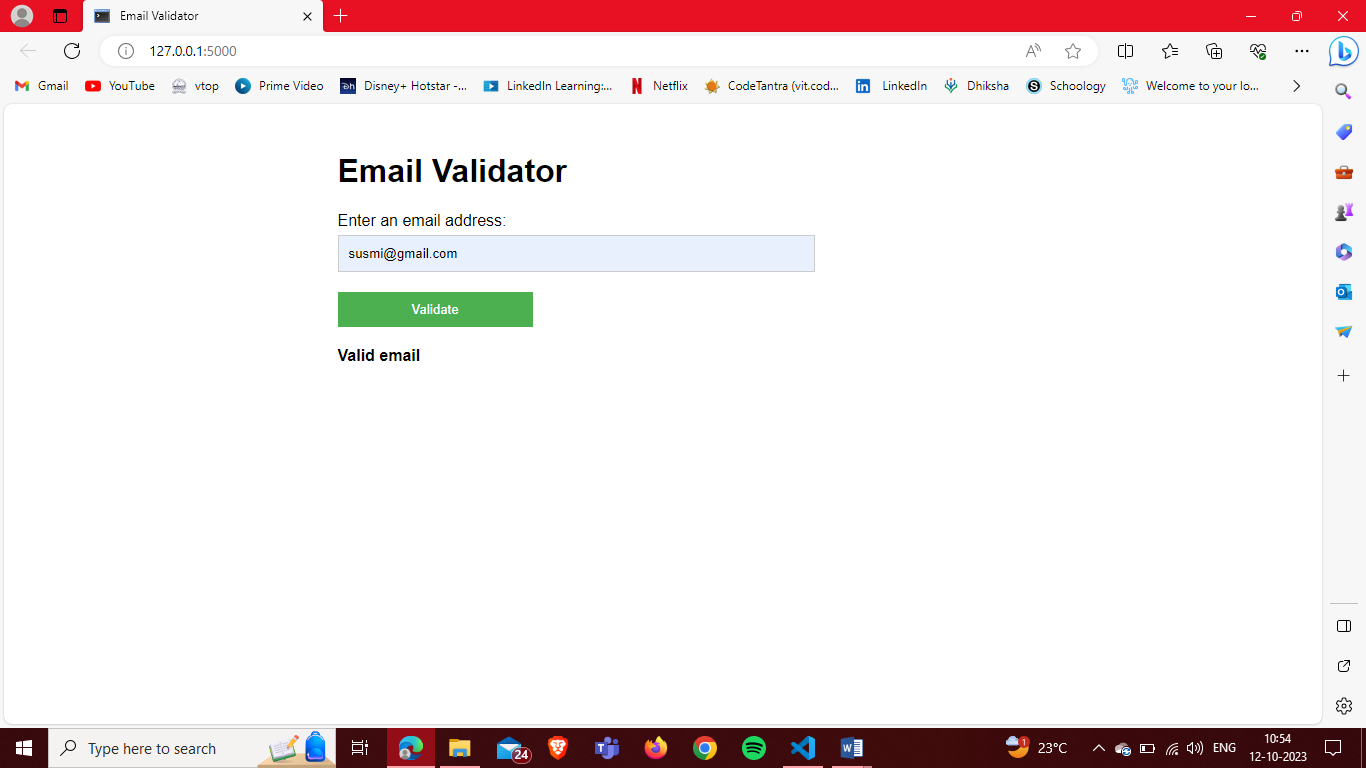
* + A **<label>** is associated with an input field for the email address.
  + An **<input>** field with type "text" allows users to enter an email address.
  + A **<button>** allows users to submit the form for email validation.

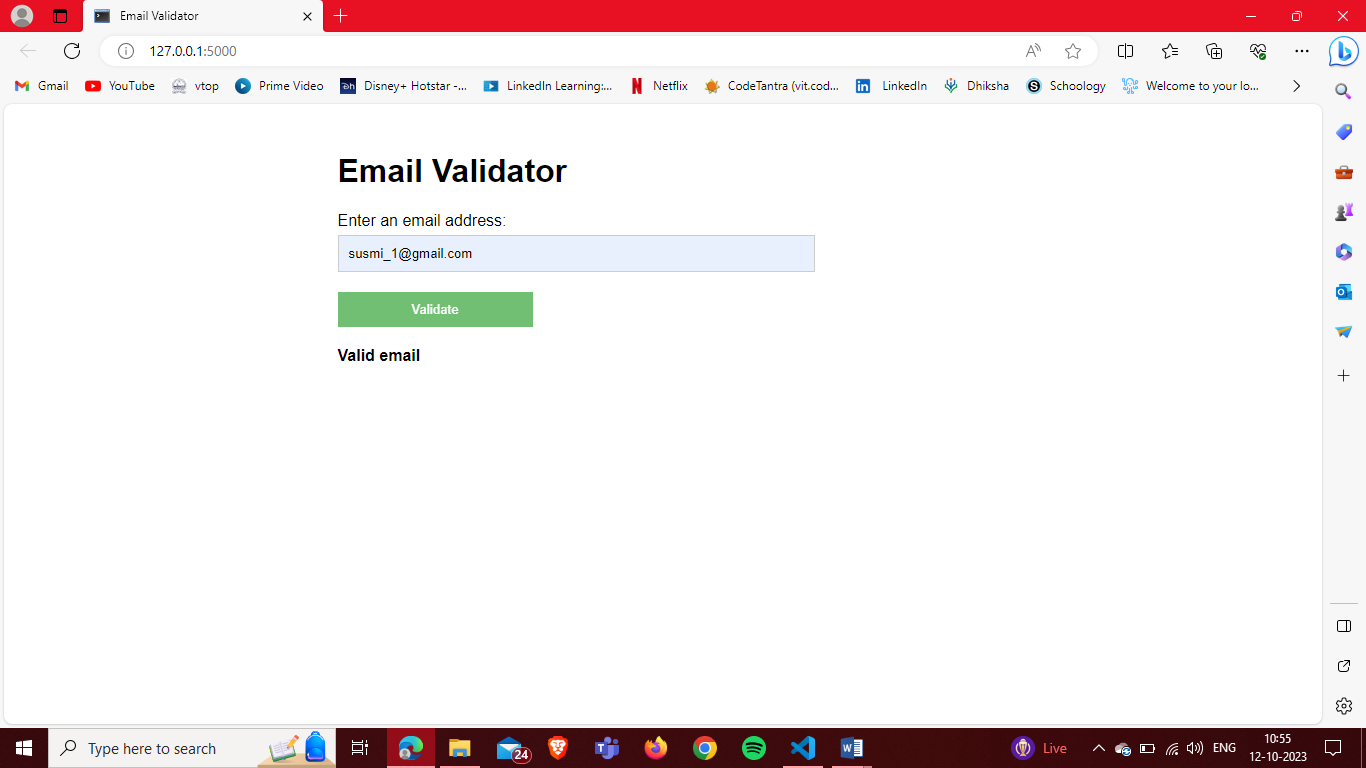
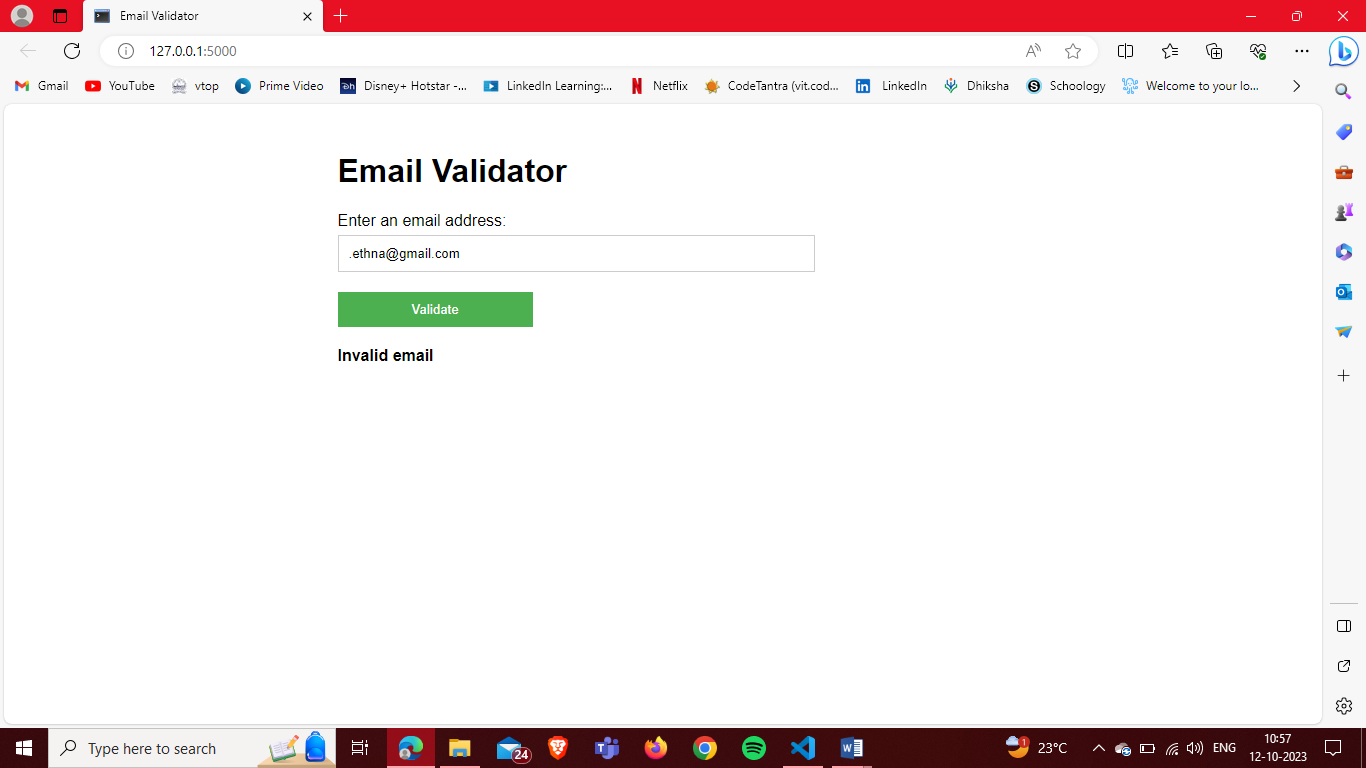
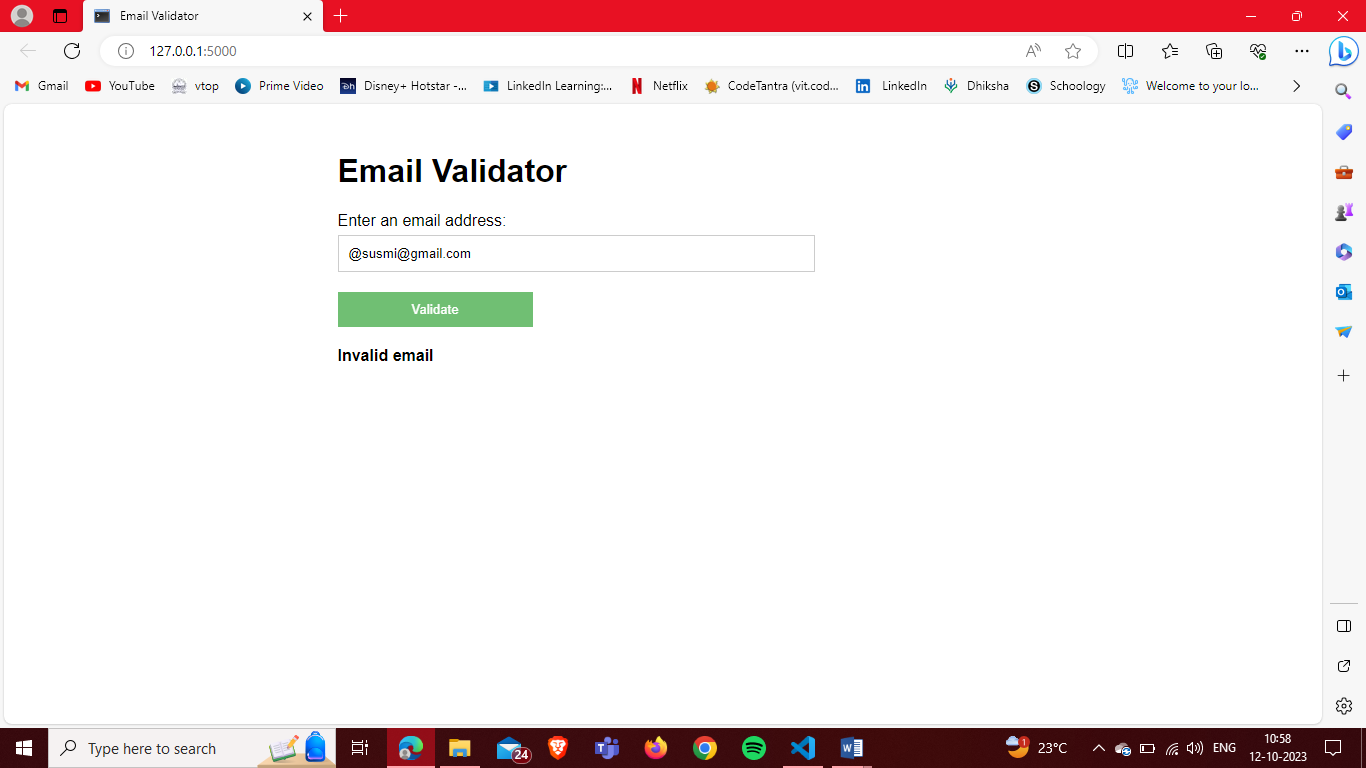
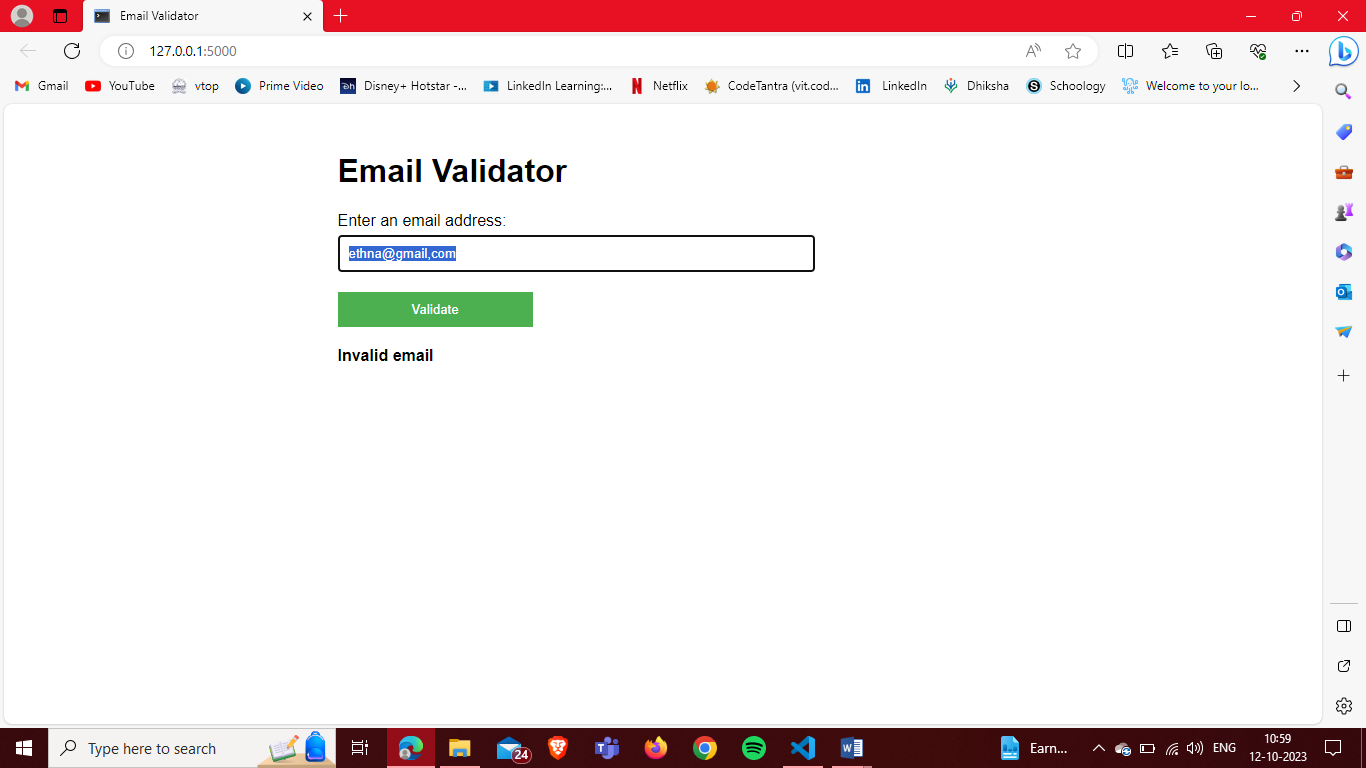
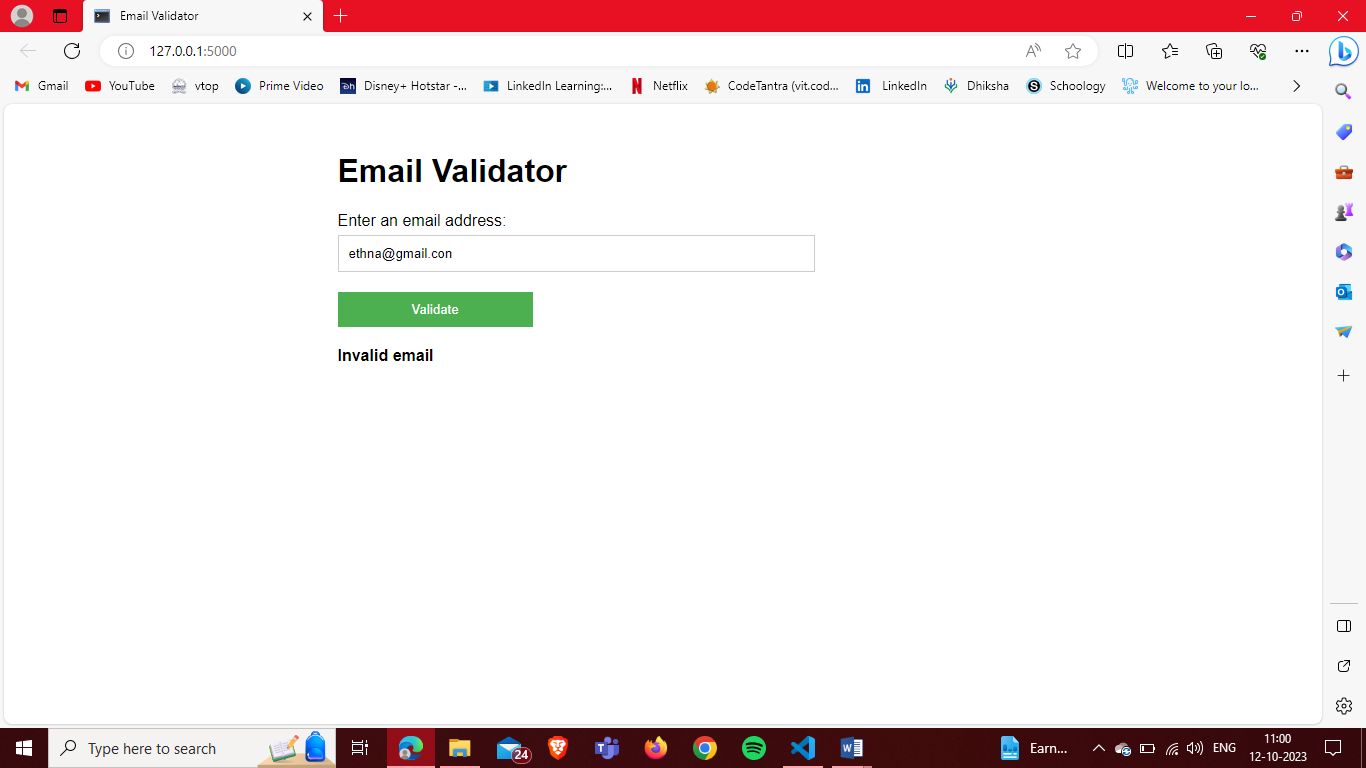
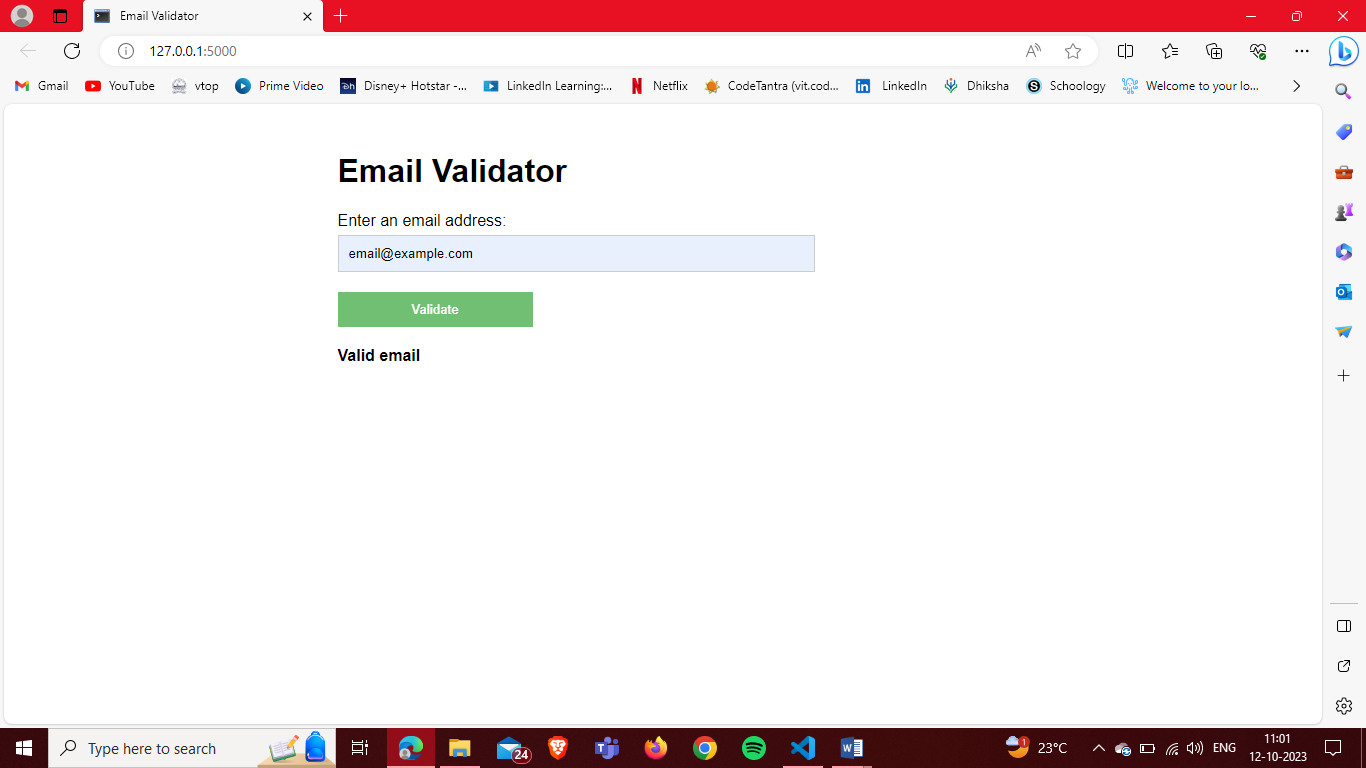
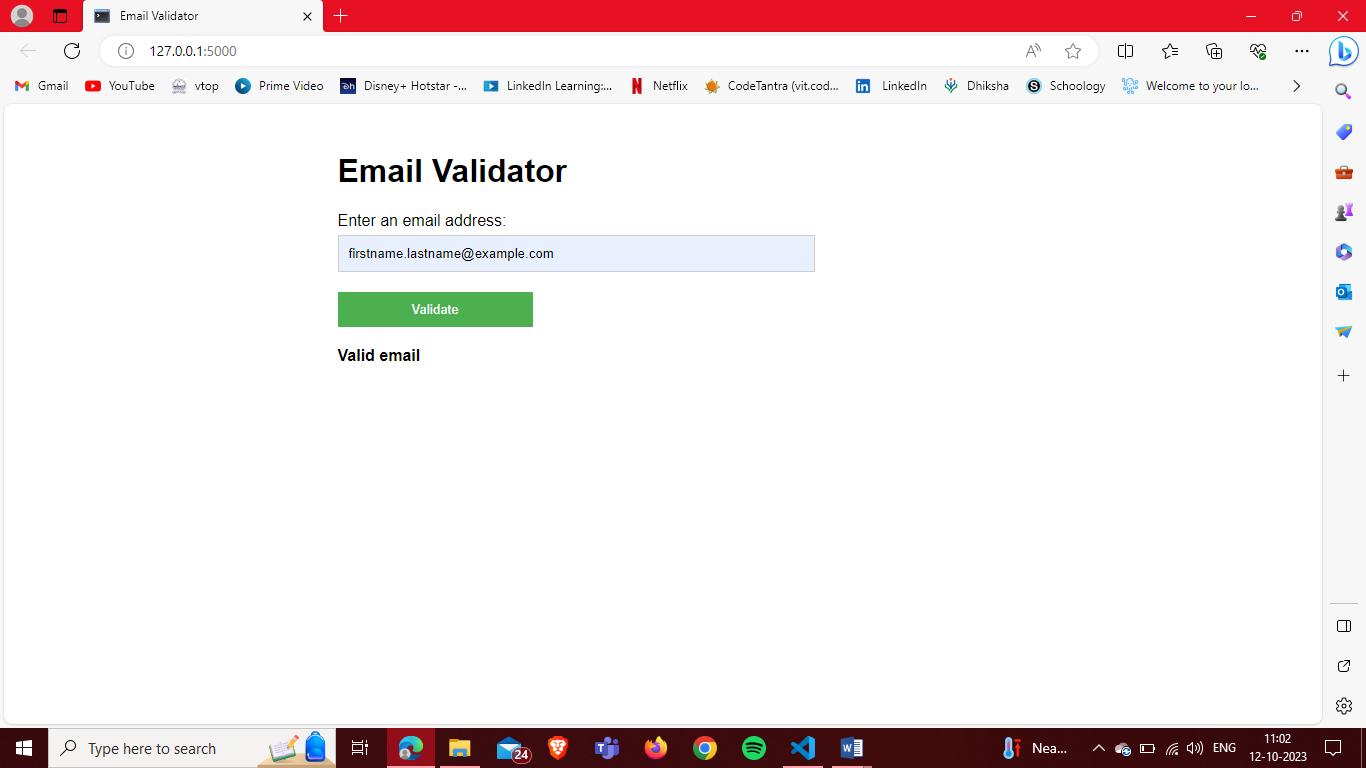
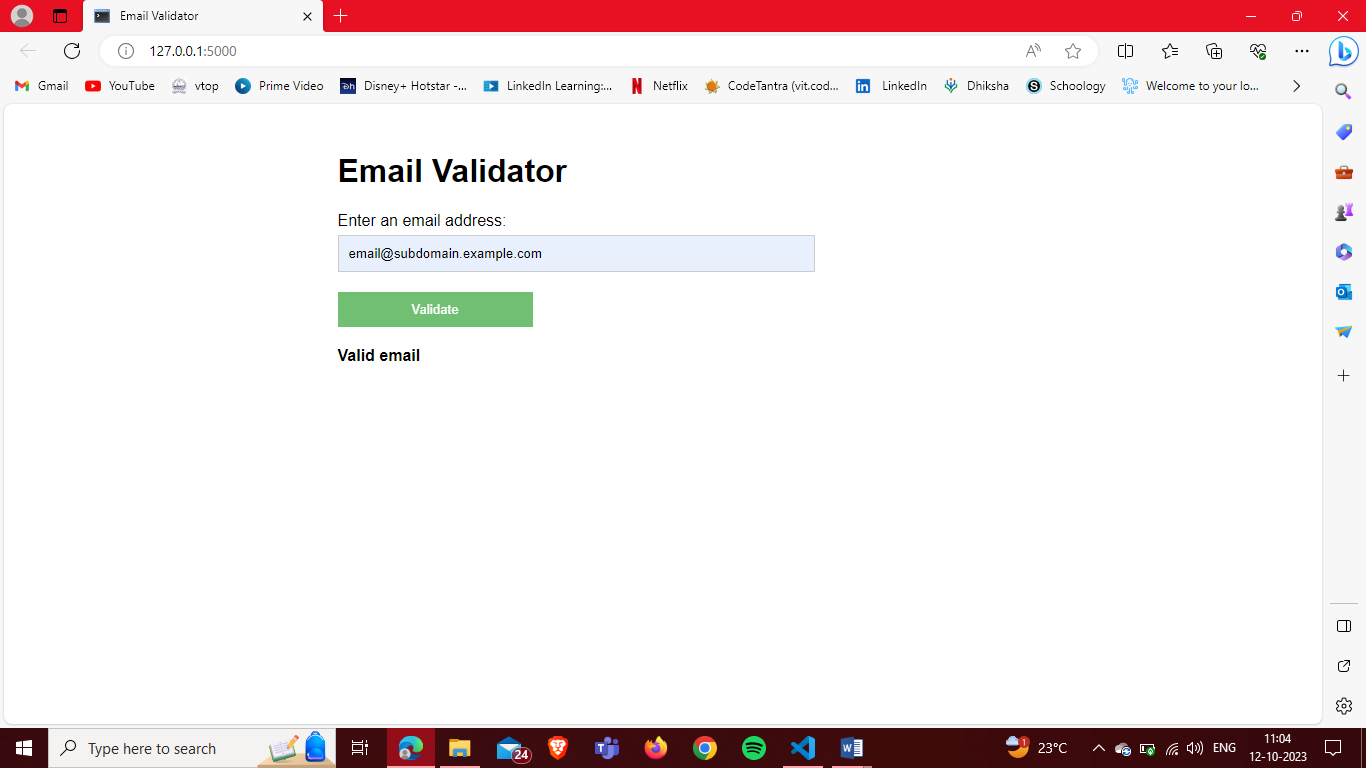
1. **JavaScript for Form Submission and Displaying Results**:

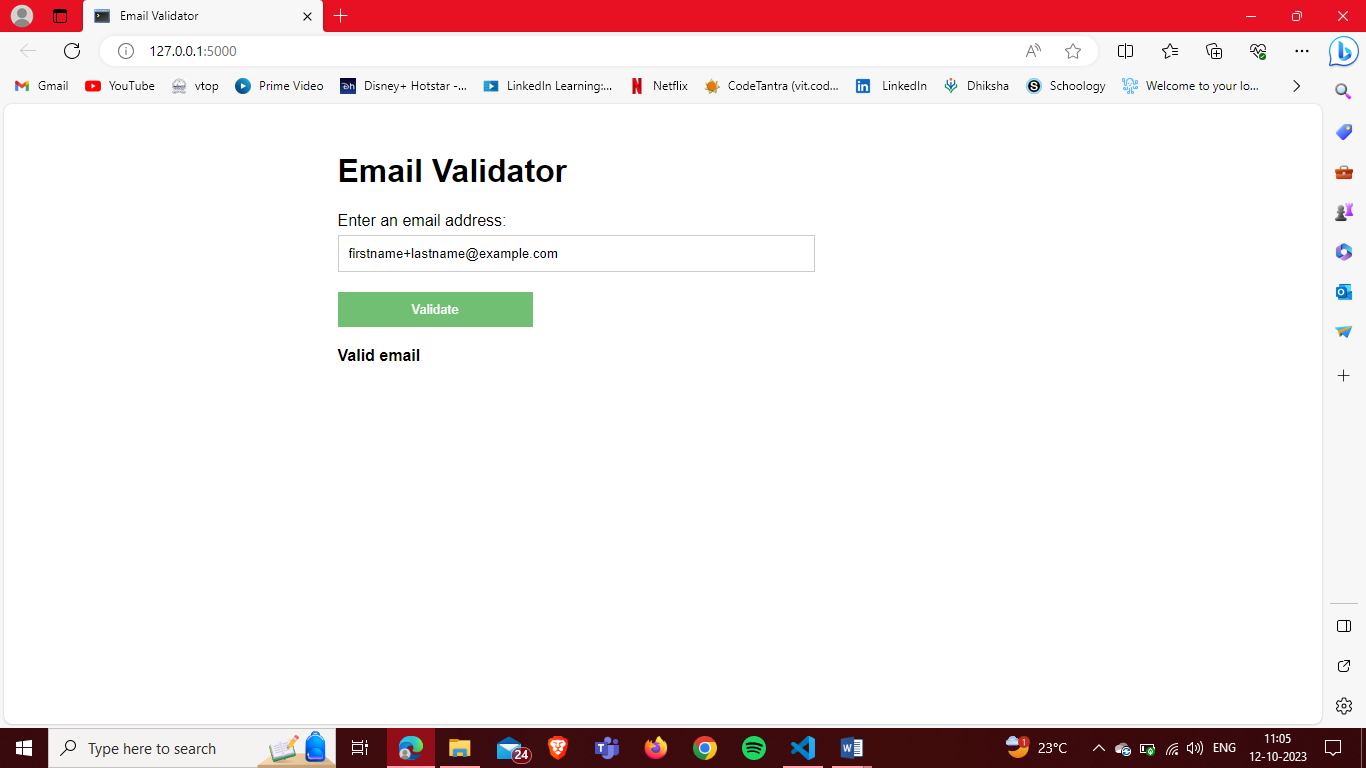
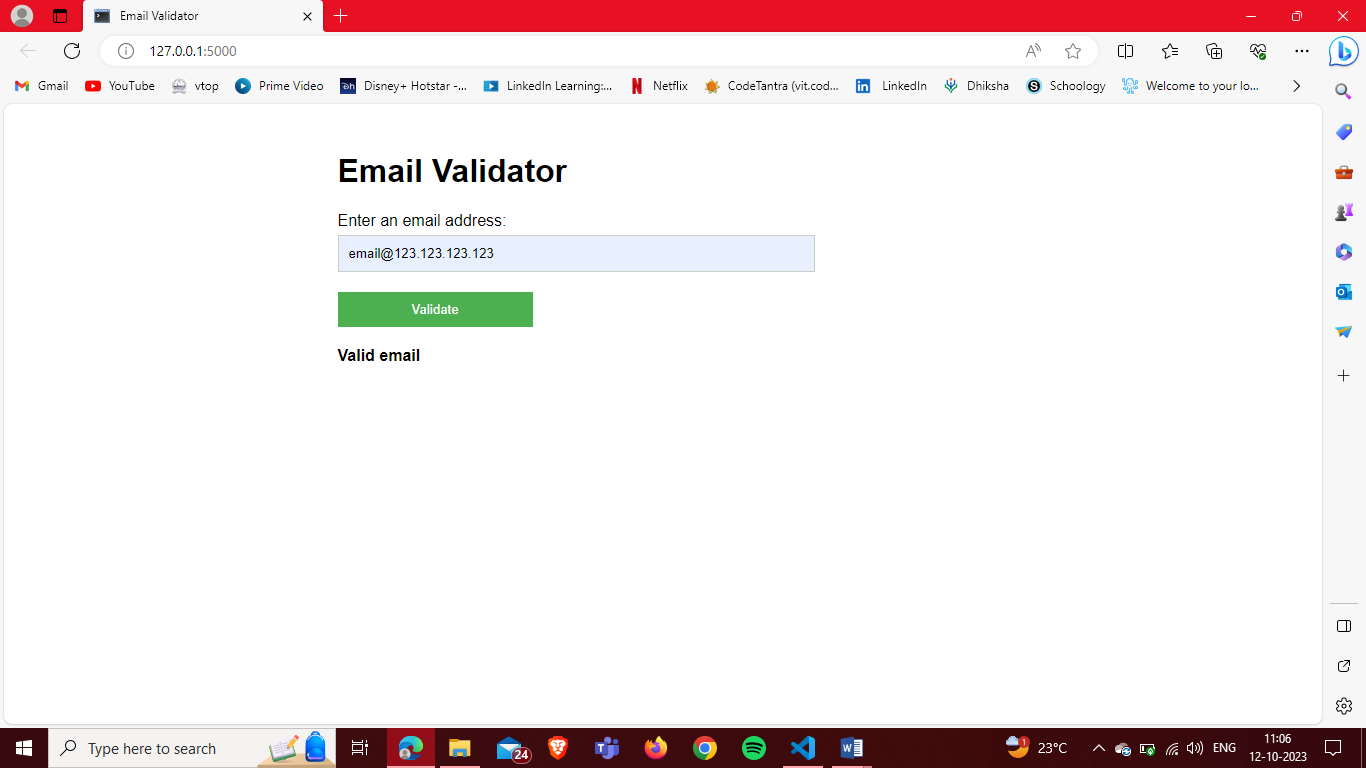
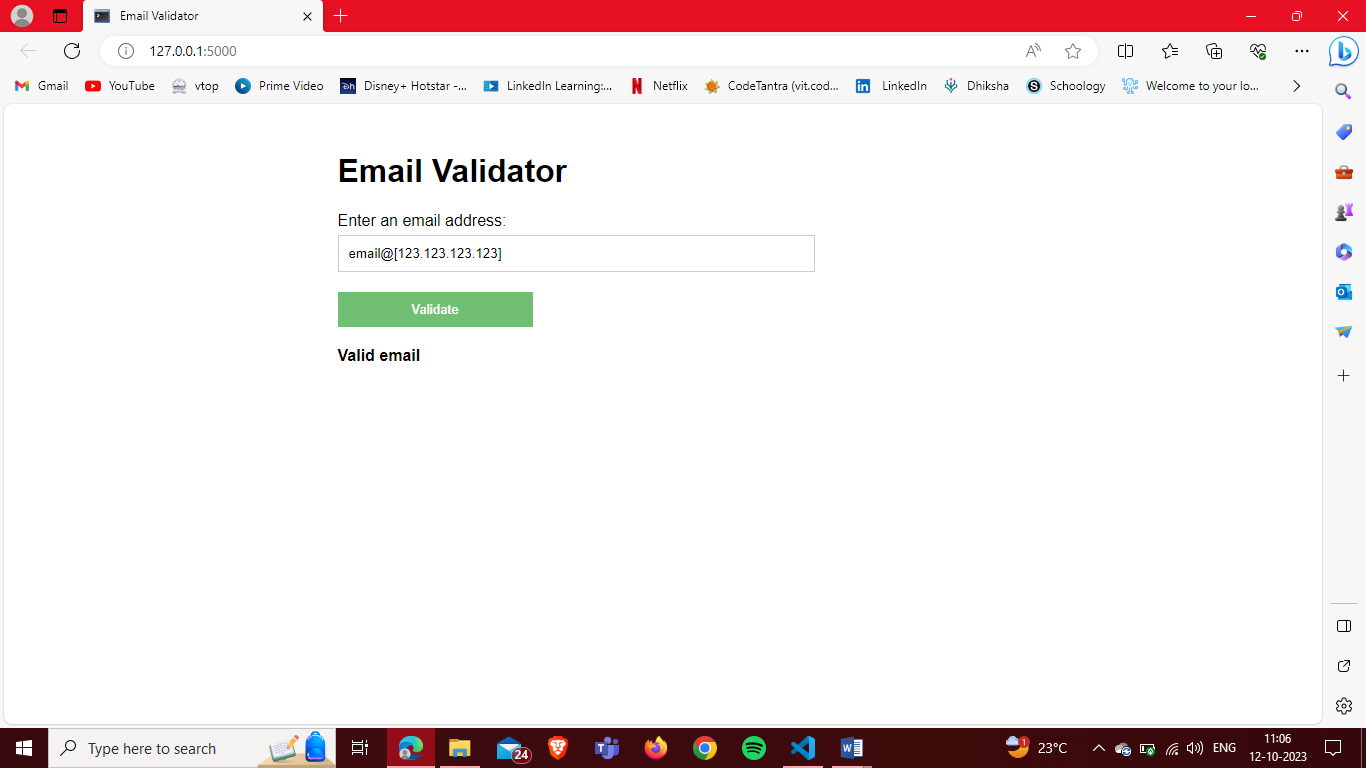
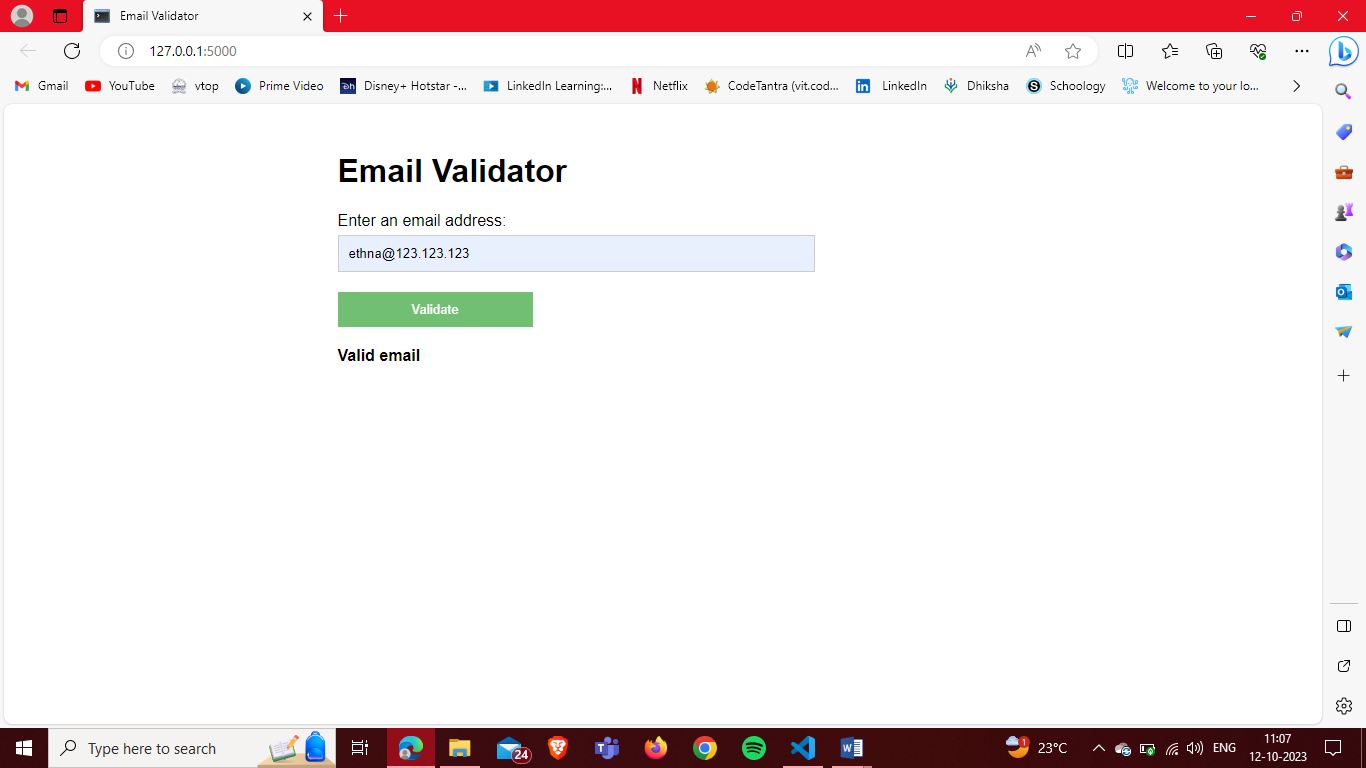
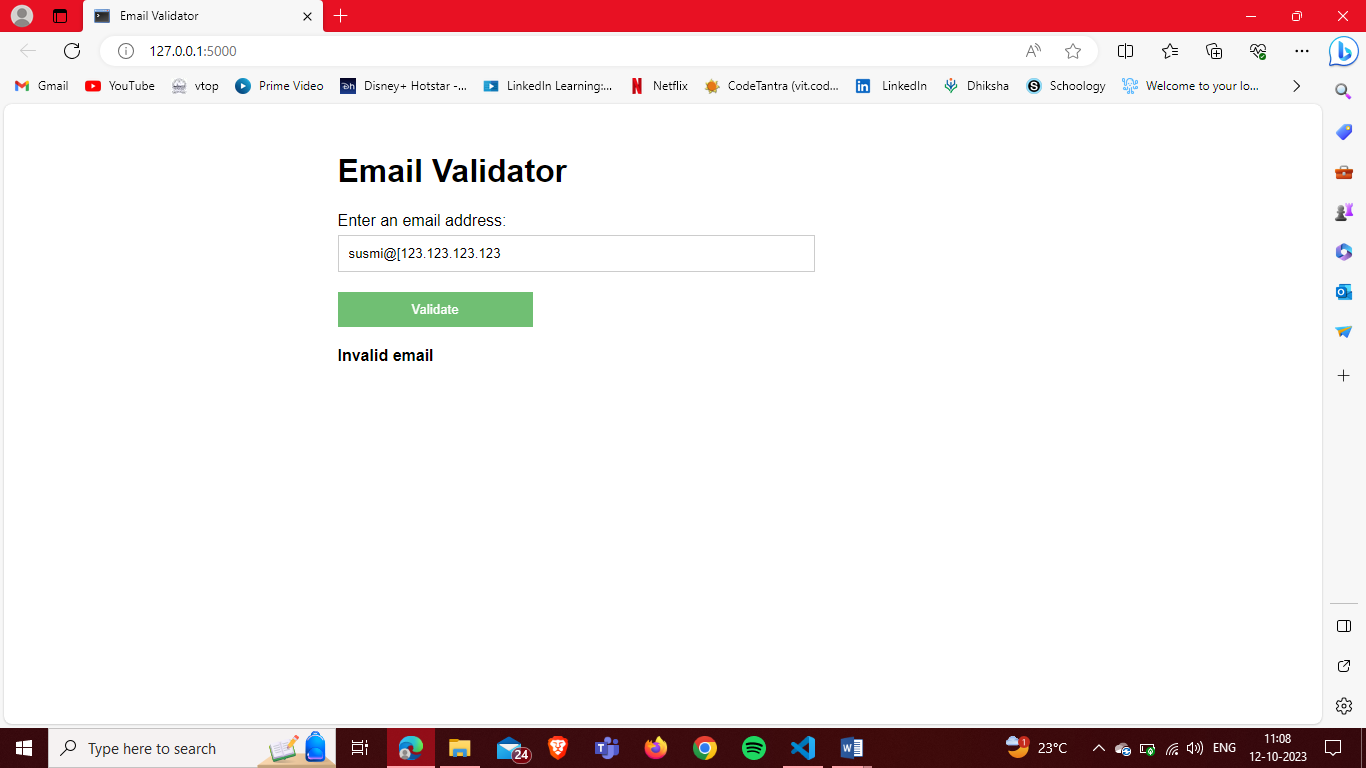
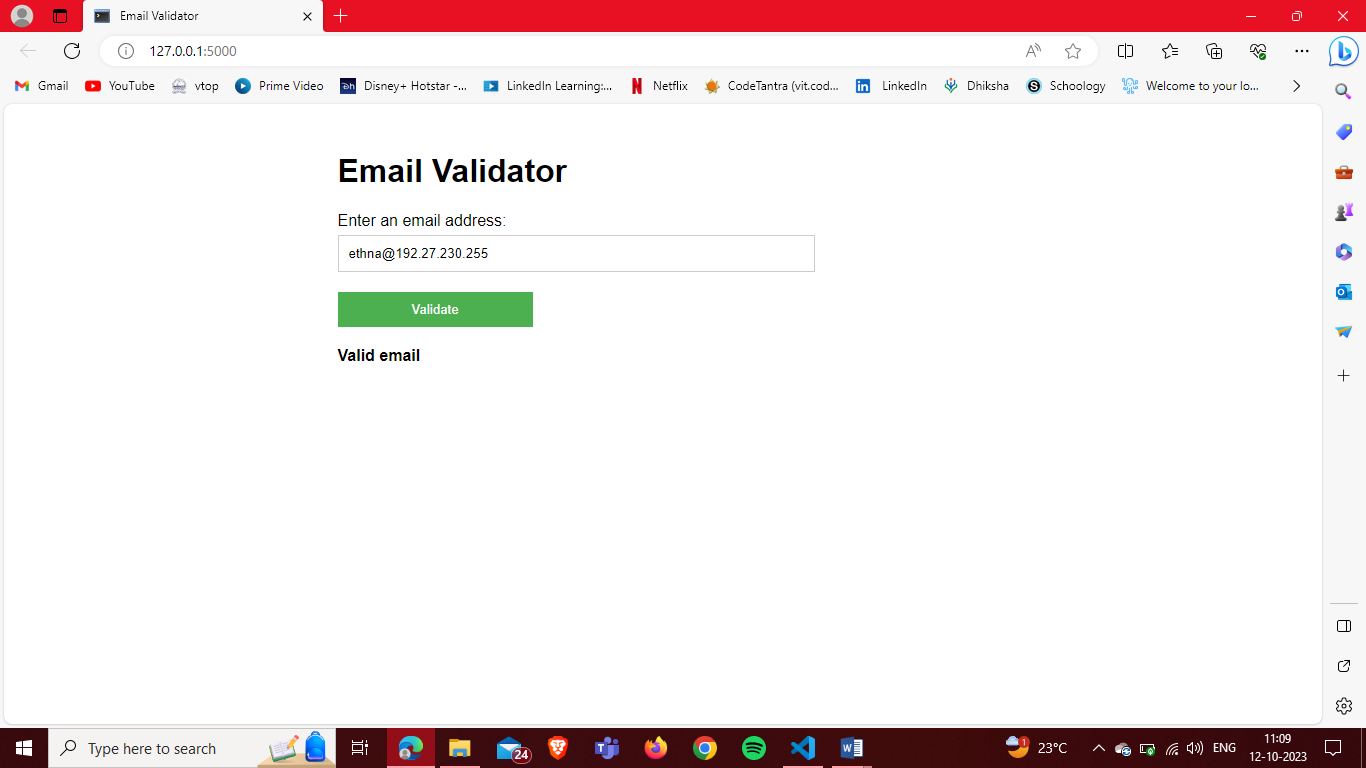
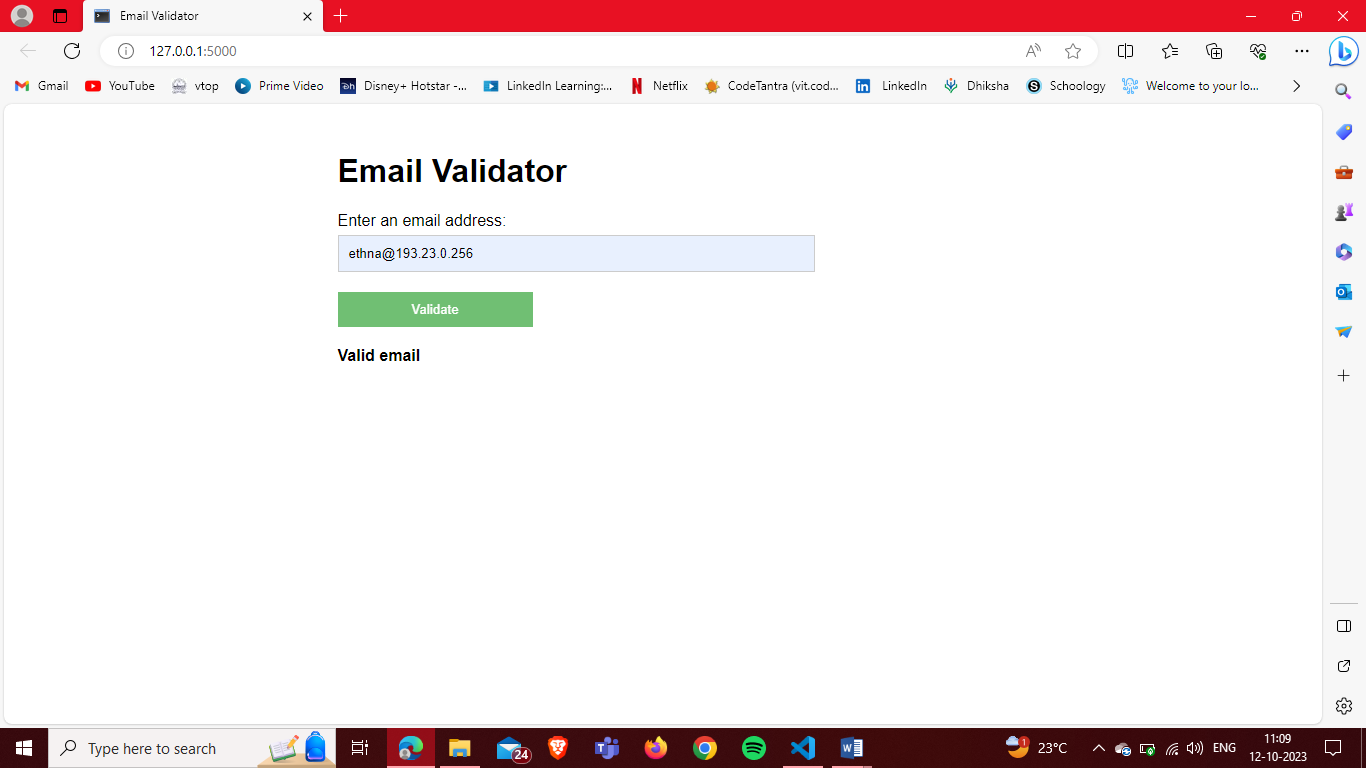
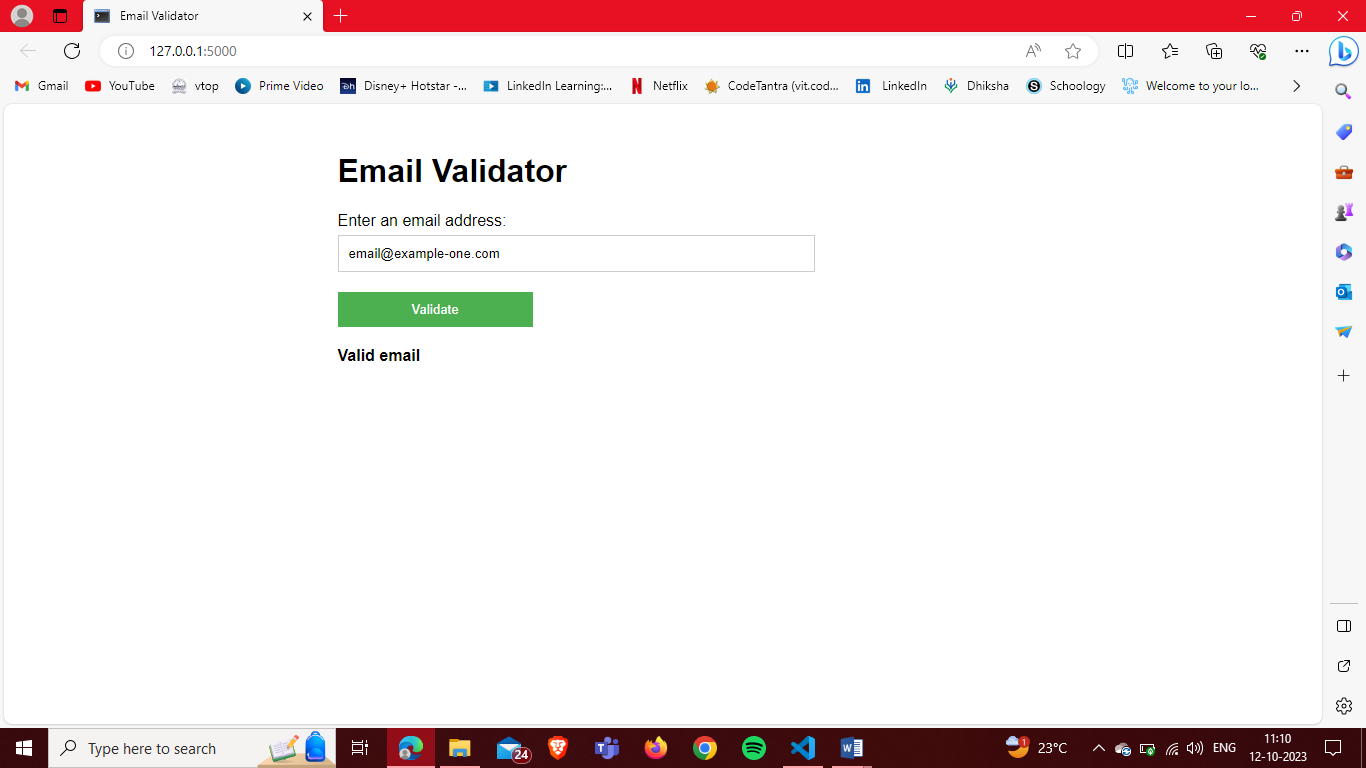
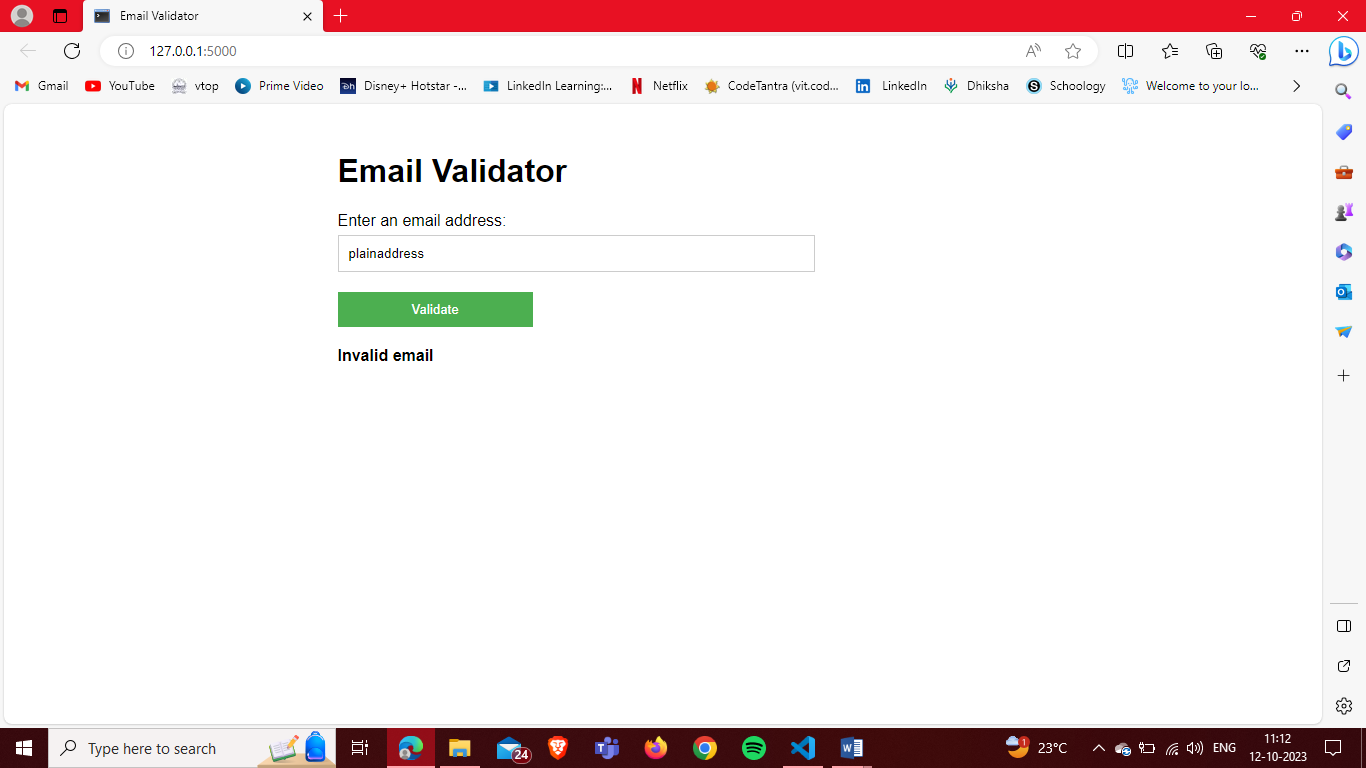
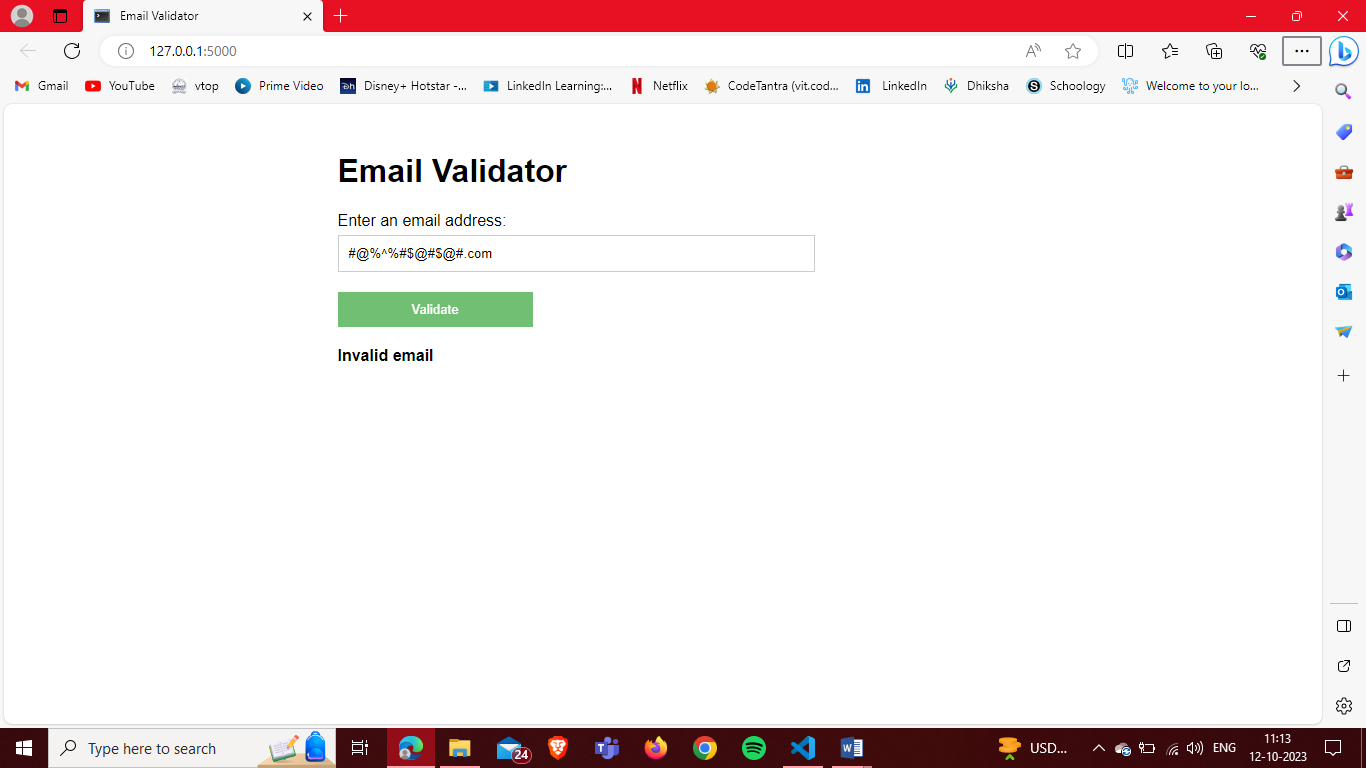
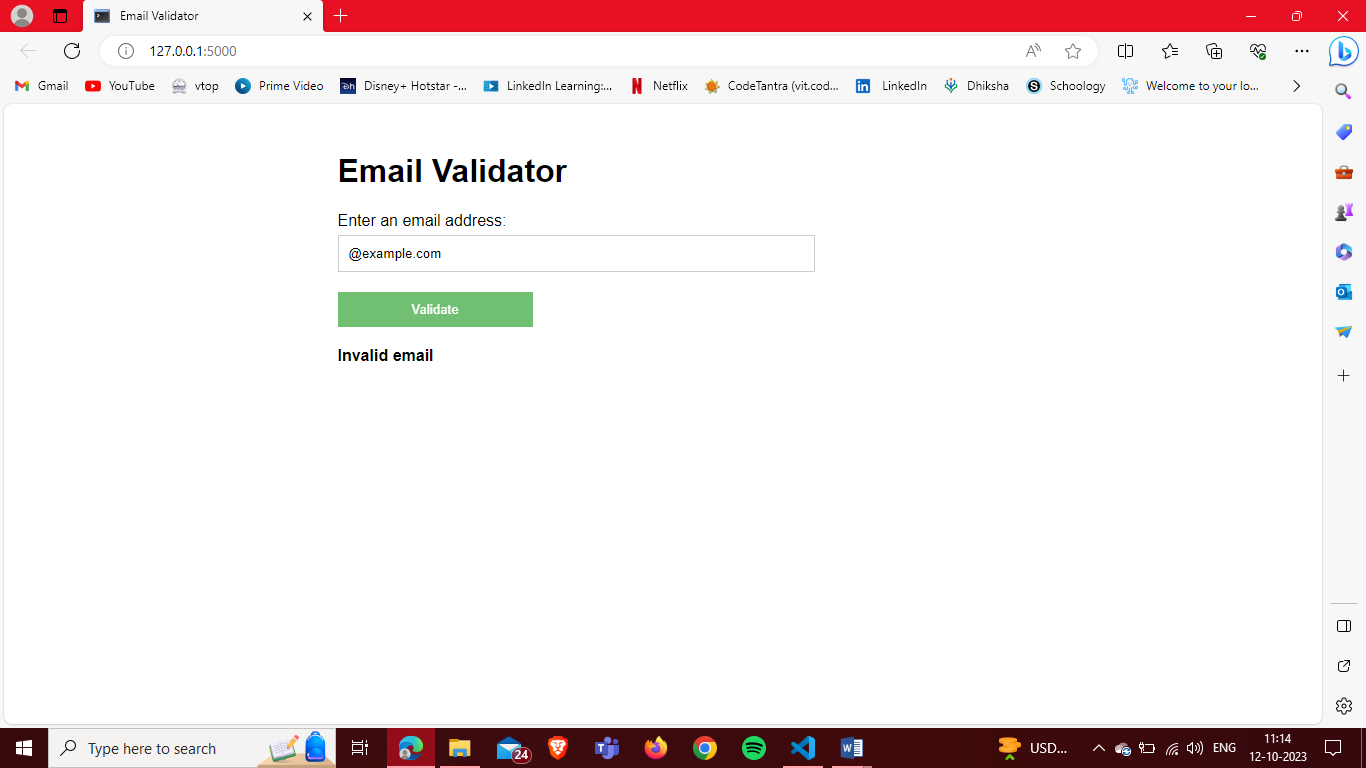
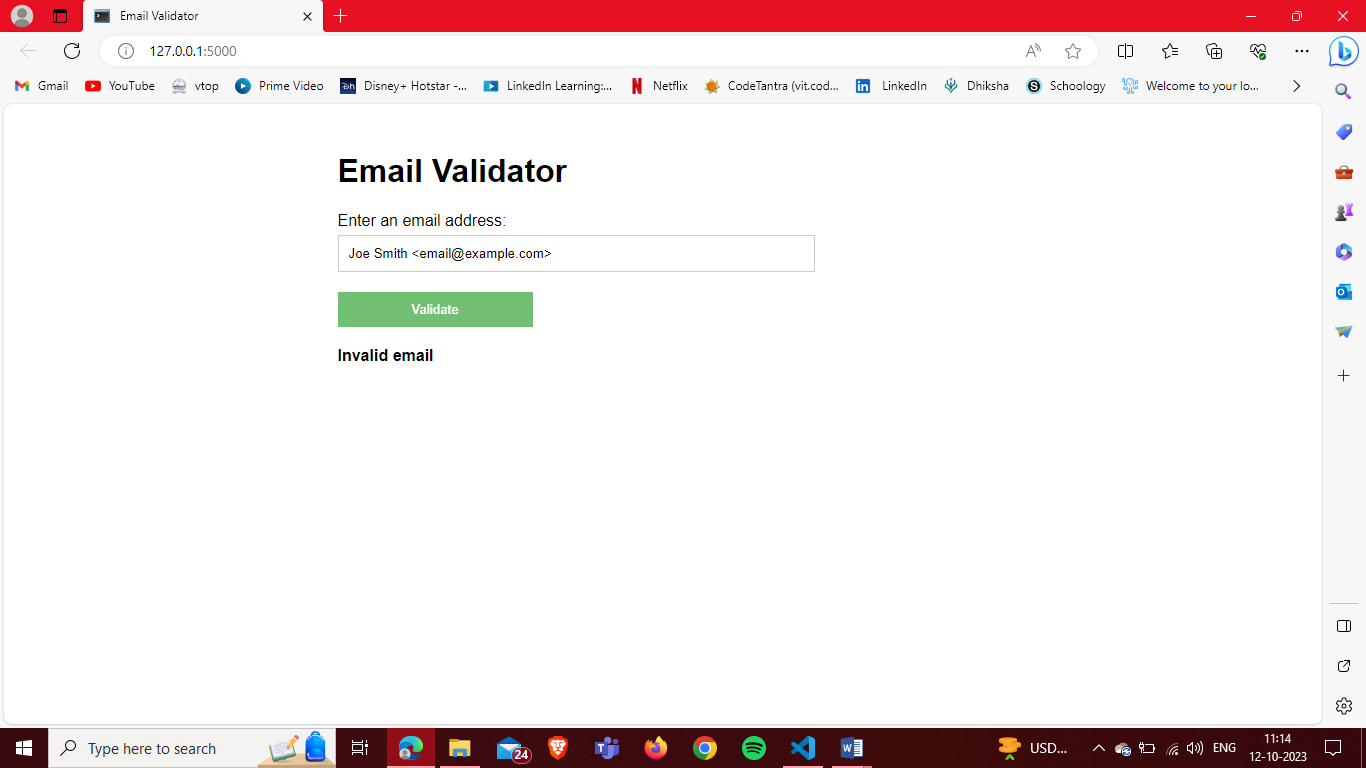
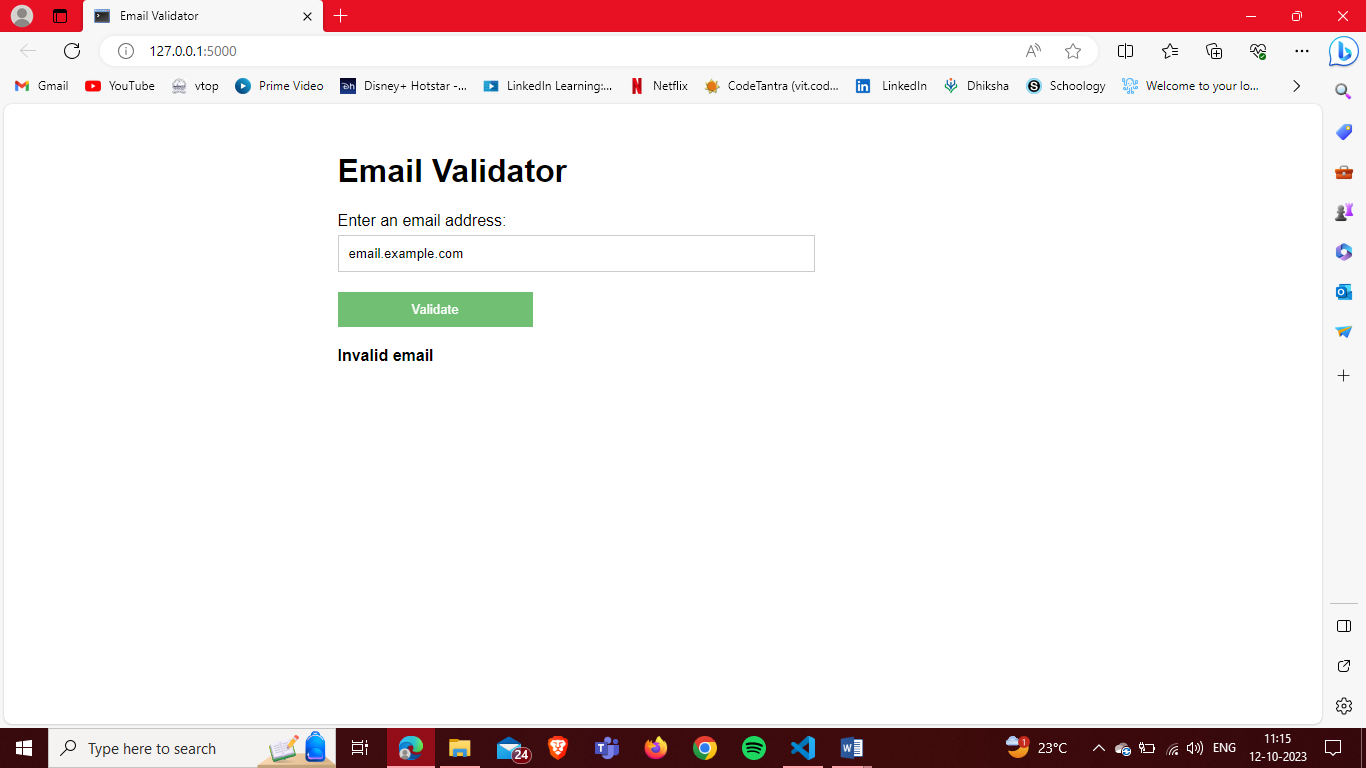
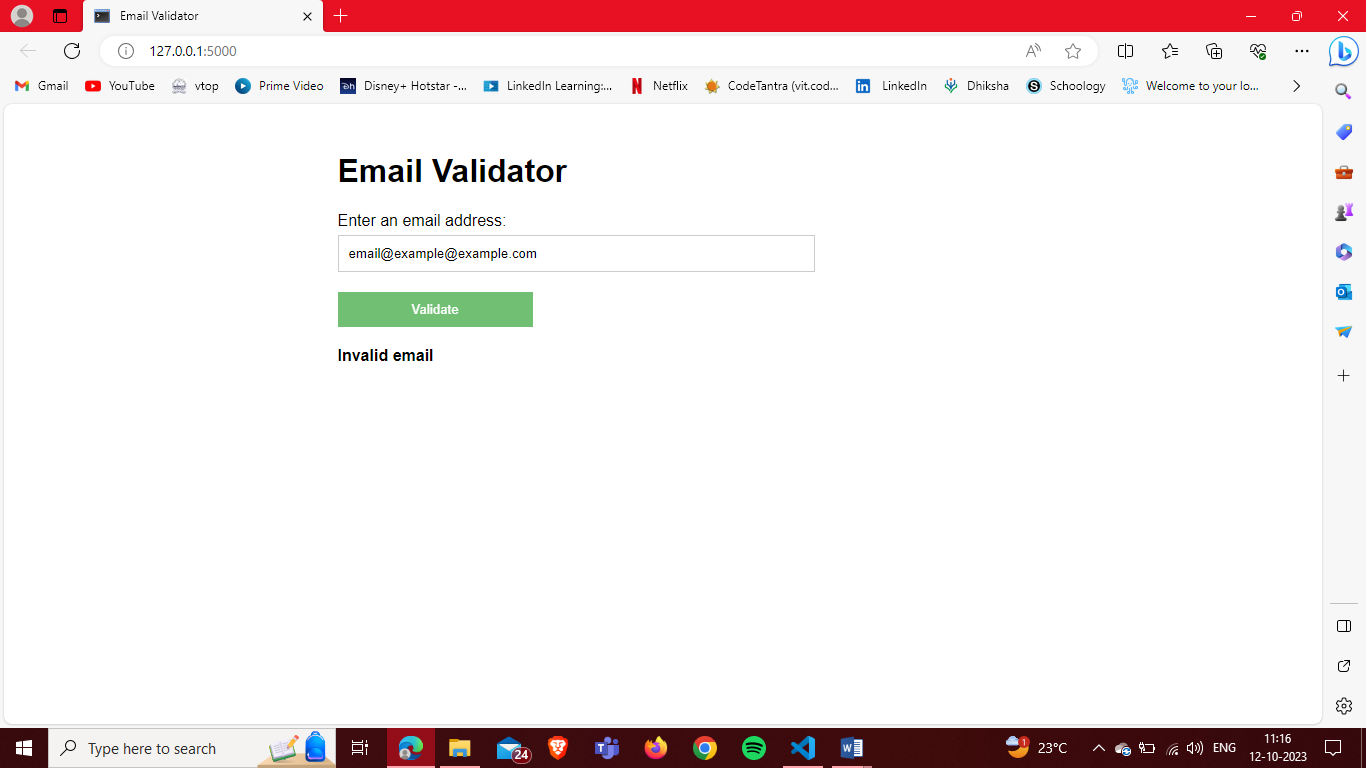
<script> ... (JavaScript code) </script>

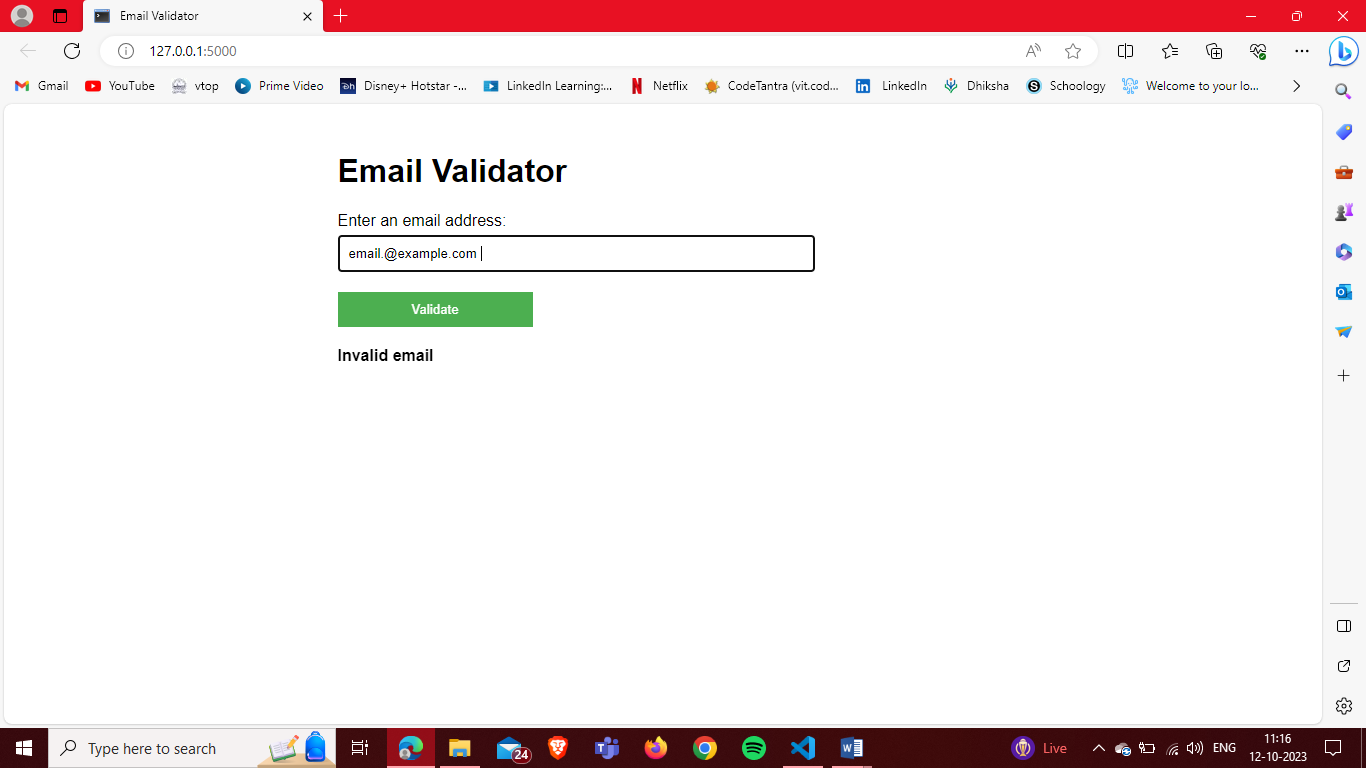
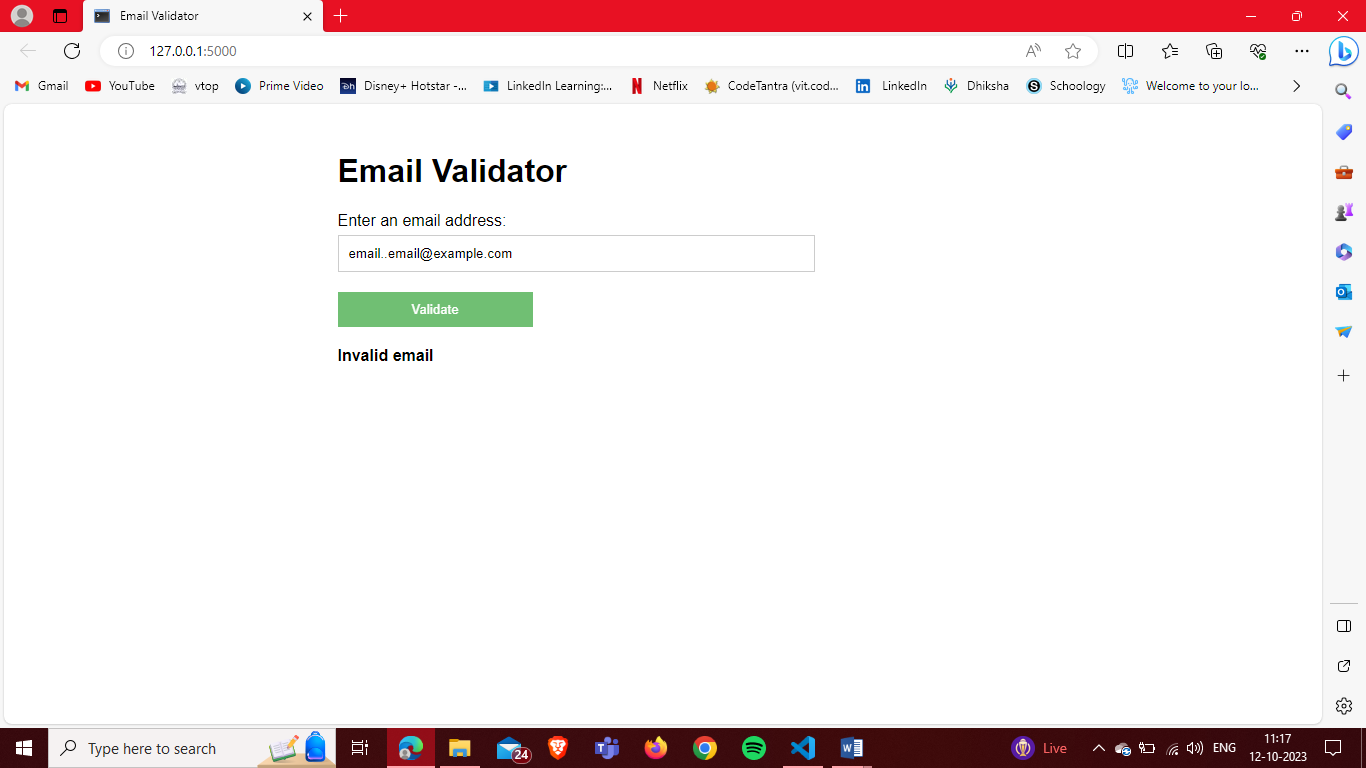
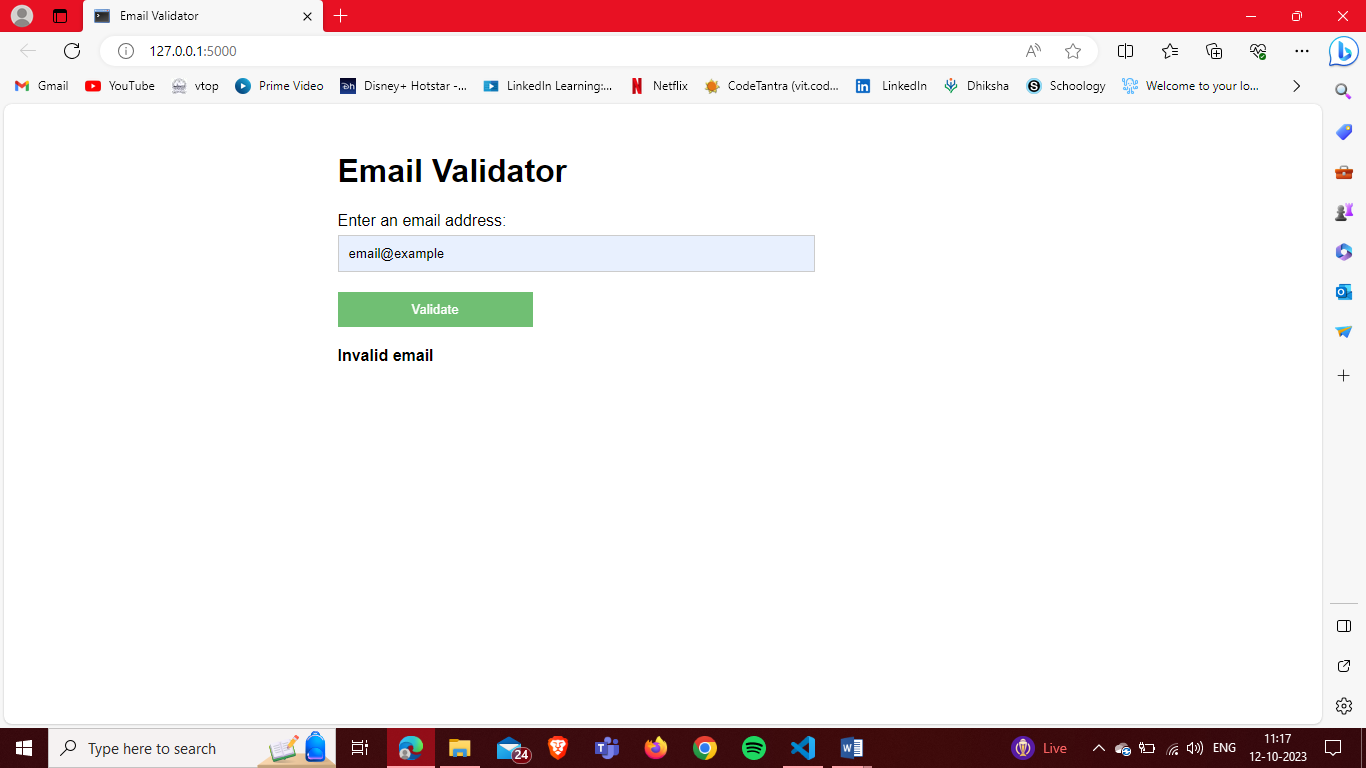
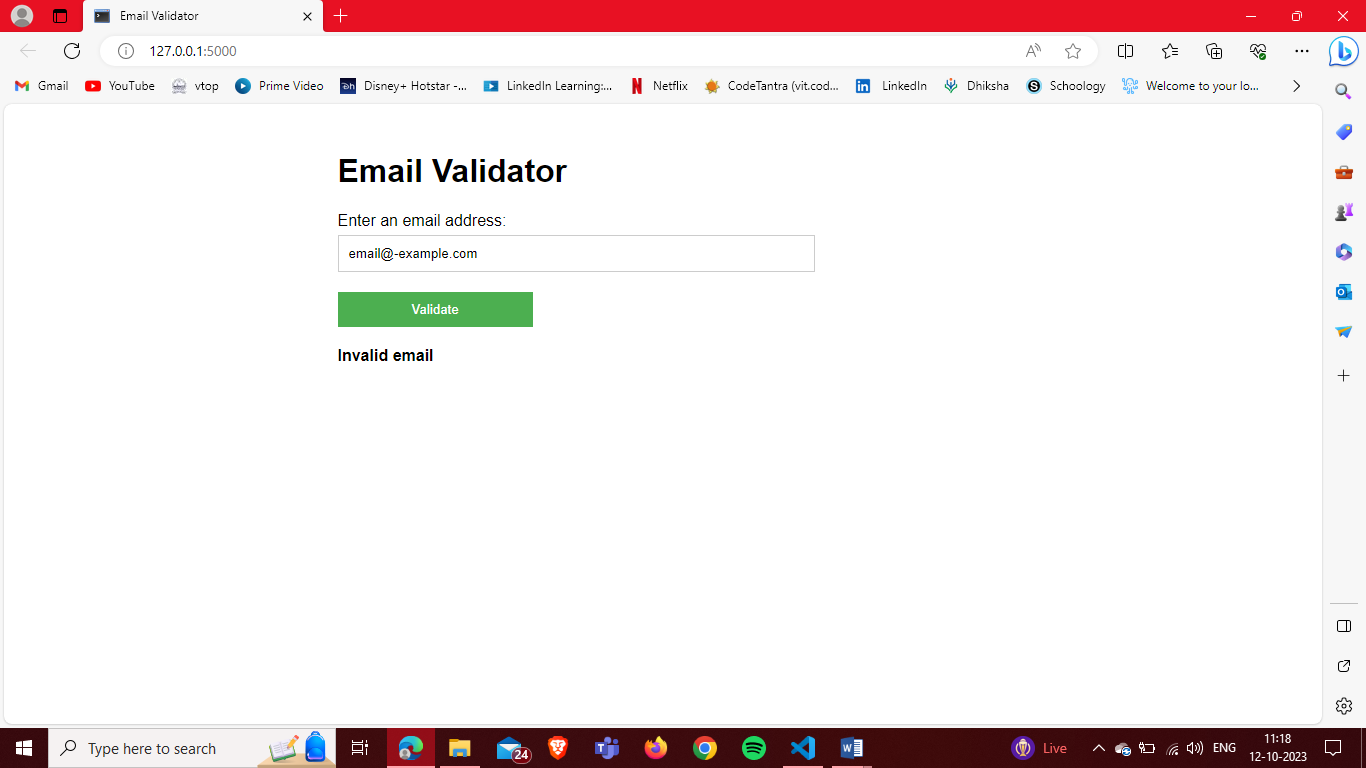
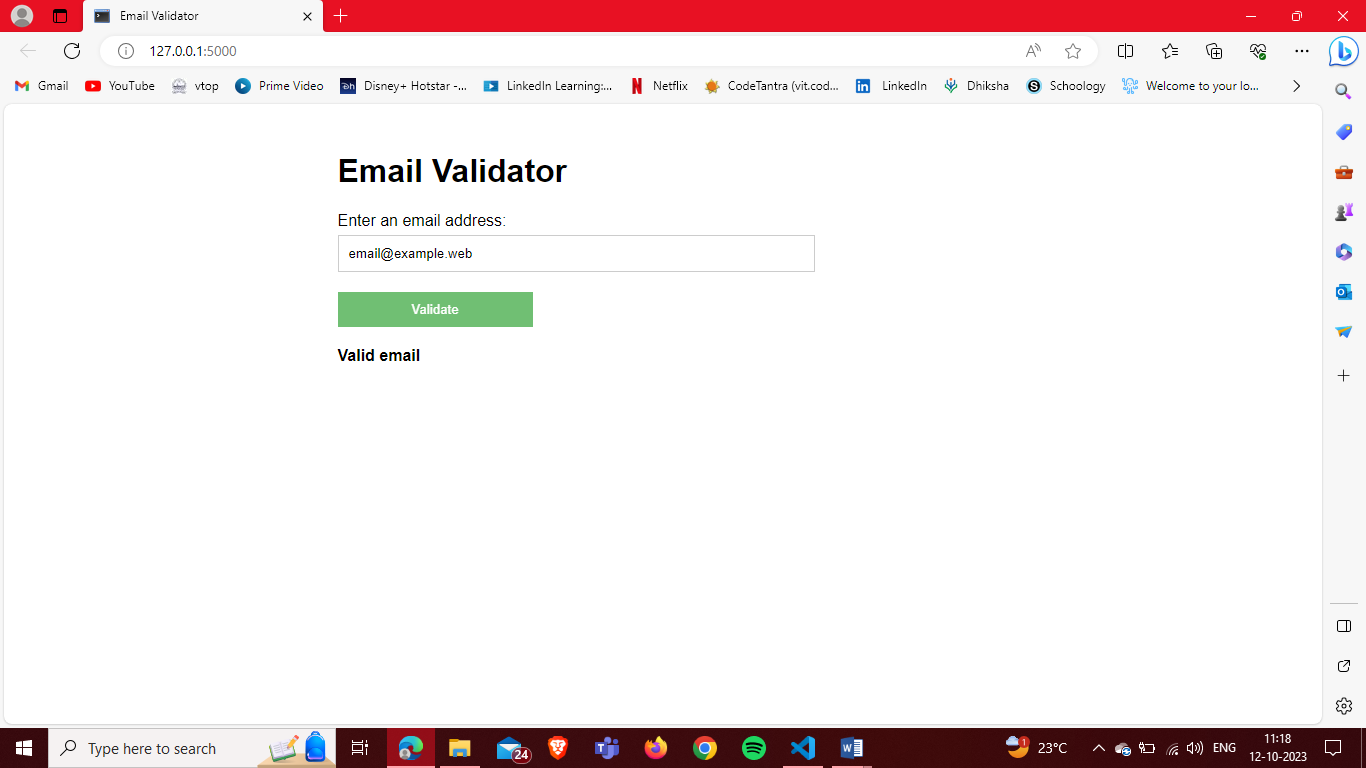
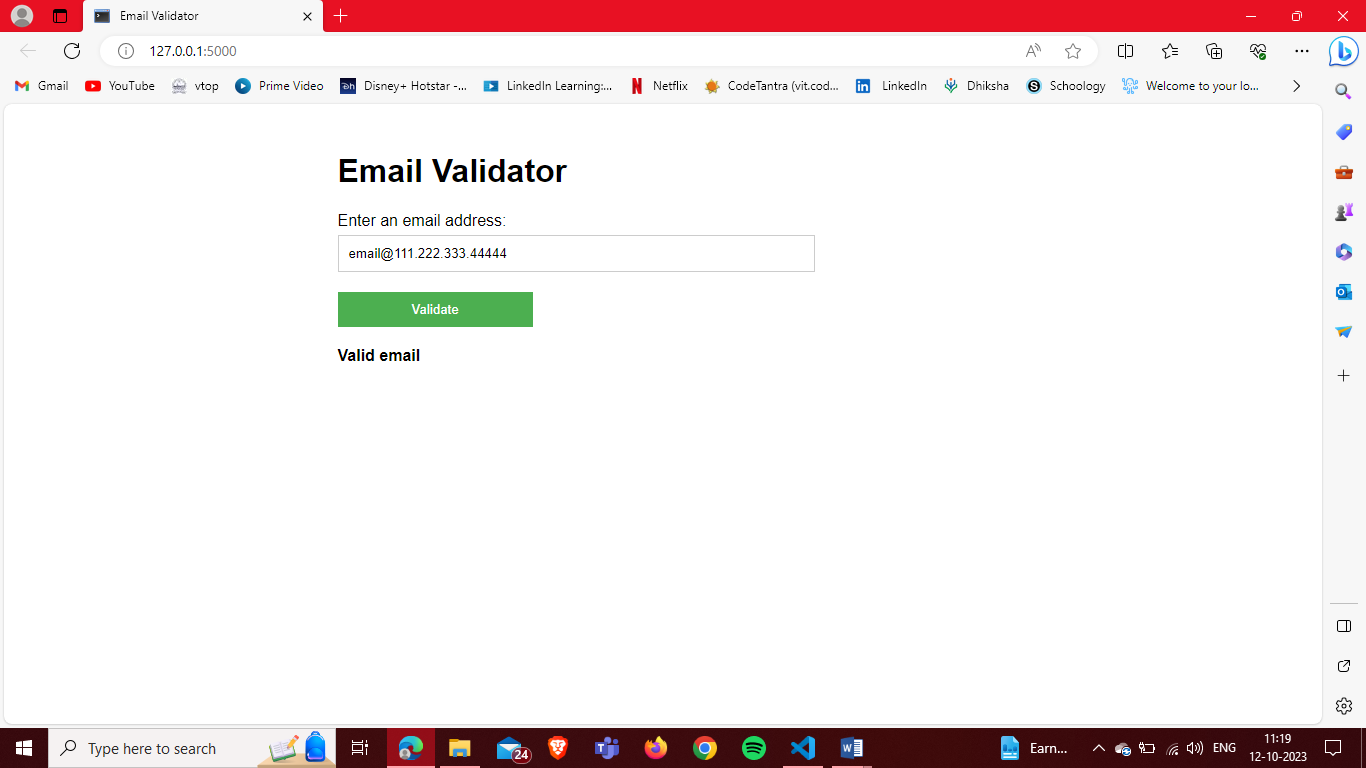
* + JavaScript code handles form submission, sends a POST request to the server, and displays the validation result on the webpage.

The JavaScript code within the **<script>** block listens for the form submission event, prevents the default form submission behaviour, retrieves the entered email, sends a POST request to the server (using **fetch**), and updates the webpage with the validation result.

**Screenshots of Testing  
Input:**[ethna@gmail.com](mailto:ethna@gmail.com)  
**Output:**  
**Input:**[susmi@gmail.com](mailto:susmi@gmail.com)  
**Output:**

**Input:**[susmi\_1@gmail.com](mailto:susmi_1@gmail.com)  
**Output:**  
  
**Input:**[.ethna@gmail.com](mailto:.ethna@gmail.com)  
**Output:** **Input:**@susmi@gmail.com  
**Output:** **Input:**ethna@gmail,com  
**Output:** **Input:**[ethna@gmail.con](mailto:ethna@gmail.con)  
**Output:** **Input:**[email@example.com](mailto:email@example.com)  
**Output:** **Input:**[firstname.lastname@example.com](mailto:firstname.lastname@example.com)  
**Output:**  
**Input:**[email@subdomain.example.com](mailto:email@subdomain.example.com)  
**Output:**

**Input:**[firstname+lastname@example.com](mailto:firstname%2Blastname@example.com)  
**Output:** **Input:**[e](mailto:email@123.123.123.123)[mail@123.123.123.123](mailto:mail@123.123.123.123)  
**Output:** **Input:**email@[123.123.123.123]  
**Output:** **Input:**[ethna@123.123.123](mailto:ethna@123.123.123)  
**Output:** **Input:**susmi@[123.123.123.123  
**Output:** **Input:**ethna@192.27.230.255  
**Output:** **Input:**[ethna@193.23.0.256](mailto:ethna@193.23.0.256)  
**Output:** **Input:**[email@example-one.com](mailto:email@example-one.com)  
**Output:** **Input:**plainaddress  
**Output:** **Input:**#@%^%#$@#$@#.com  
**Output:** **Input:**@example.com  
**Output:** **Input:**Joe Smith [<email@exa](mailto:email@example.com)m[ple.com>](mailto:email@example.com)  
**Output:** **Input:**email.example.com  
**Output:** **Input:**[email@example@example.com](mailto:email@example@example.com)  
**Output:**

**Input:**[email.@example.com](mailto:email.@example.com)  
**Output:** **Input:**[email..email@example.com](mailto:email..email@example.com)  
**Output:** **Input:**email@example  
**Output:** **Input:**[email@-example.com](mailto:email@-example.com)  
**Output:** **Input:**[email@example.web](mailto:email@example.web)  
**Output:** **Input:**[email@111.222.333.44444](mailto:email@111.222.333.44444)  
**Output:** **Input:**[email@example..com](mailto:email@example.com)  
**Output:**