21/01/2024

Telecommunication Software

Report 3

Jehanne Sarrazin

230AEM053

<u>For this whole practical exercise, you can find my work in the GitHub repository: https://github.com/Niennaaa/TelecommunicationSoftware</u>

<u>Example 1 & 2:</u>

This question was rather troubling, as it asked to do a string-related webpage (so as I understand it, string only and no design) that can combine CSS and JS.

To this end, I realized the HTML webpage that verifies both examples as one that you can find in the folder "<u>EXAMPLE1".</u>

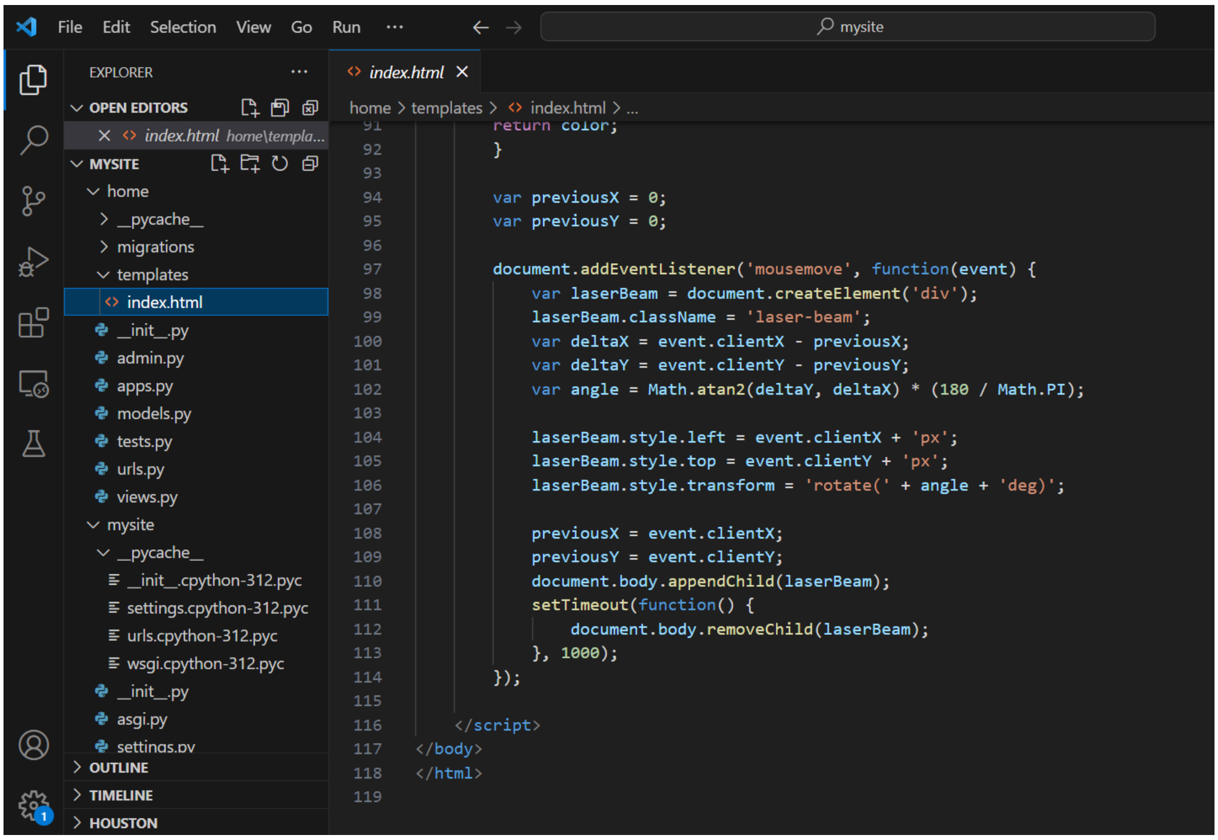
To run for project from the project folder, you can use VSCode to open the project and run the following in the terminal:

```
cd mysite
python manage.py runserver
```

And the project will be up and running at http://127.0.0.1:8000/home/

The resulting HMTL page is a simple "hello world" with changing size and color. The background can be clicked to change the background of the page's color, and a yellow line follows the user's mouse as it moves.

Here is a screenshot of the project open and showing the bit of the code responsible for the yellow line following the mouse:



```
            return color;
            }

            var previousX = 0;
            var previousY = 0;

            document.addEventListener('mousemove', function(event) {
                var laserBeam = document.createElement('div');
                laserBeam.className = 'laser-beam';
                var deltaX = event.clientX - previousX;
                var deltaY = event.clientY - previousY;
                var angle = Math.atan2(deltaY, deltaX) * (180 / Math.PI);

                laserBeam.style.left = event.clientX + 'px';
                laserBeam.style.top = event.clientY + 'px';
                laserBeam.style.transform = 'rotate(' + angle + 'deg)';

                previousX = event.clientX;
                previousY = event.clientY;
                document.body.appendChild(laserBeam);
                setTimeout(function() {
                    document.body.removeChild(laserBeam);
                }, 1000);
            });

        </script>
    </body>
    </html>
```

I tried to make it so that the line would also follow the line in its angle but the result is rather poor, although I personally thought it was neat.

Example 3:

I built the cloud message board using PythonAnywhere as host for the cloud service. You can find the website for my message board here : https://niennaa.pythonanywhere.com/messages1/message-board/

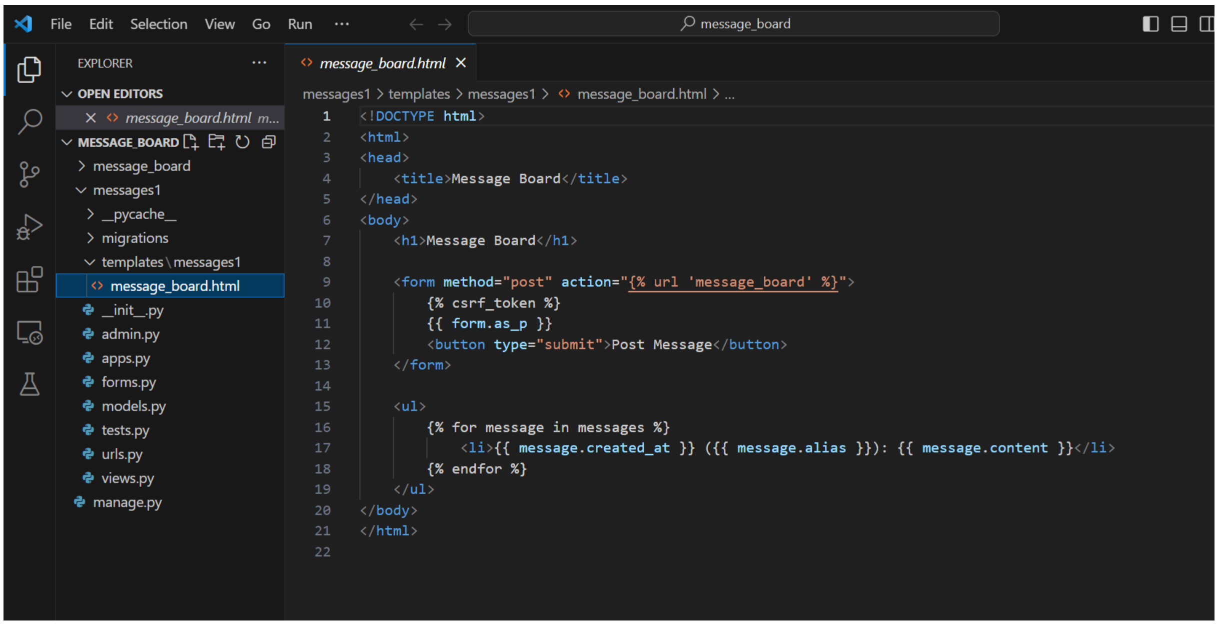
You can also open the project in the folder “message_board” and run it in the same way as the previous example. Note that since the actual database for the messages is ran on PythonAnywhere, you will not find the messages that are on the online platform when running the project locally.

For now, it’s a wall you can post on using a username and I haven’t implemented a service of destination to a specific user yet as there is no real service of authentication required. This would mean simply adding a system of filter so that when one identifies as a certain username, they can view all the messages that implemented said username as destinator. It would be like a system of tag on each message.

For actual authentication I believe tools such as Keycloak would be more adapted to this kind of platform (a message board) when looking to create an actual secure website.

Additionally my platform obviously lack design for now as I’ve been running out of time to style it, but you can take my example 1 & 2 as example of what can be implemented.

Here is a screenshot of the template that serves as the message board :



```html
<!DOCTYPE html>
<html>
<head>
    <title>Message Board</title>
</head>
<body>
    <h1>Message Board</h1>

    <form method="post" action="{% url 'message_board' %}">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Post Message</button>
    </form>

    <ul>
        {% for message in messages %}
            <li>{{ message.created_at }} ({{ message.alias }}): {{ message.content }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

As you can see it’s pretty short, as it only contains the basics in the building of the platform. I was focused on trying to keep it simple to make sure there wouldn’t be any issue in its integration online.

Example 4:

I tried to run Django's different response types on my side but after hours of trying to make them work without success, I moved on and decided to simply explain here what they are and what results I was expecting from all of them.

- **HttpResponse class:**

It represents an HTTP response sent back to the client. It namely has ten subclasses:

HttpResponseRedirect: Takes in the URL to which the user must be redirected, status code 302

HttpResponsePermanentRedirect: Same but for a permanent redirect, status code 301.

HttpResponseNotModified: The page will not be modified since the user's last visit. Status code 304.

HttpResponseBadRequest: The request was invalid. Status code 400.

HttpResponseNotFound: The request wasn't found. Status code 404.

HttpResponseForbidden: The request was found but the server is refusing the access to it. Status code 403.

HttpResponseNotAllowed: Must have either GET or POST in argument. The method wasn't allowed by the server. 405 status code.

HttpResponseGone: The requested resource isn't available anymore and has no redirection. 410 status code.

HttpResponseServerError: The server had an error and can't execute the request. 500 status code

HttpResponse can also be **customized** depending on the need in development.

- **JsonResponse class:**

Return a JSON encoded response.

- **StreamingHttpResponse class:**

Streams a response from Django to the browser. Used to generate long responses that would usually be cut by Django.

- **FileResponse class:**

A subclass of StreamingHttpResponse, optimized for binary files.