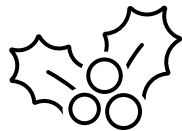


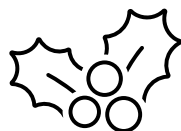
Juliane Follereau

230AMM020



Telecommunication Software

Practical work 3



November 2023

Contents

Information	1
Example 1	2
Basic	2
More funny	3
Christmas theme	5
Example 2	8
Return an HTML page	8
Example 3	10
Classic messaging	10
Santa claus mail	16
Example 4	23

Information

In this practical work I wanted to have more fun than in the others since in the instructions it was noted that this exercise was more flexible. So I chose a theme in the realization of my tasks which is the Christmas theme!! He will come back to some of my examples.

At the start of each new example I will briefly explain what I did so that you can understand it better. Then for each achievement I would give certain code, the architecture of the Django project as well as the final rendering on the internet page accompanied by its url.

I did everything on VSCode which seemed more practical to me than python for using Django but my opinion is personal.

Here are the beginning steps for each of the "new django projects" that I followed each time:

- Install django directly on python then see what version it is. Here we can see that I have version 4.2.7. I would like to point out that I had a lot of problems installing django, I had to in addition to having anaconda navigator install python on its own for it to work as I wanted.

```
C:\Users\julia>cd C:\Users\julia\OneDrive\Documents\A4\RTU\Telecommunications Software\christmas

C:\Users\julia\OneDrive\Documents\A4\RTU\Telecommunications Software\christmas>python -m pip install Django
Requirement already satisfied: Django in c:\users\julia\appdata\local\programs\python\python312\lib\site-packages (4.2.7)
Requirement already satisfied: asgiref<4,>=3.6.0 in c:\users\julia\appdata\local\programs\python\python312\lib\site-packages (from Django) (3.7.2)
Requirement already satisfied: sqlparse>=0.3.1 in c:\users\julia\appdata\local\programs\python\python312\lib\site-packages (from Django) (0.4.4)
Requirement already satisfied: tzdata in c:\users\julia\appdata\local\programs\python\python312\lib\site-packages (from Django) (2023.3)

C:\Users\julia\OneDrive\Documents\A4\RTU\Telecommunications Software\christmas>python -m django --version
4.2.7
```

- After all that it is only a matter of starting a new project directly at the desired location and then going inside:

```
C:\Users\julia\OneDrive\Documents\A4\RTU\Telecommunications Software\christmas>django-admin startproject merrychristmas1
```

```
C:\Users\julia\OneDrive\Documents\A4\RTU\Telecommunications Software\christmas>cd merrychristmas1
```

- And the last step is of course to create a new application directly in the project initialized just before. This is the file where our model, our work will be.

```
C:\Users\julia\OneDrive\Documents\A4\RTU\Telecommunications Software\christmas\merrychristmas1>python manage.py startapp bigpresent
```

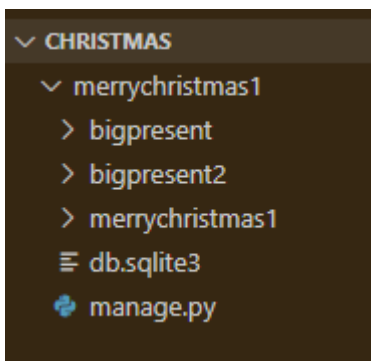
After all these steps which I repeated several times depending on what I wanted to do we can start using django and these many features.

Example 1

In this first example I achieved three things: a first which basically meets the instructions, a second with more fantasy and an html code and a last with even more fantasy and this time on the Christmas theme.

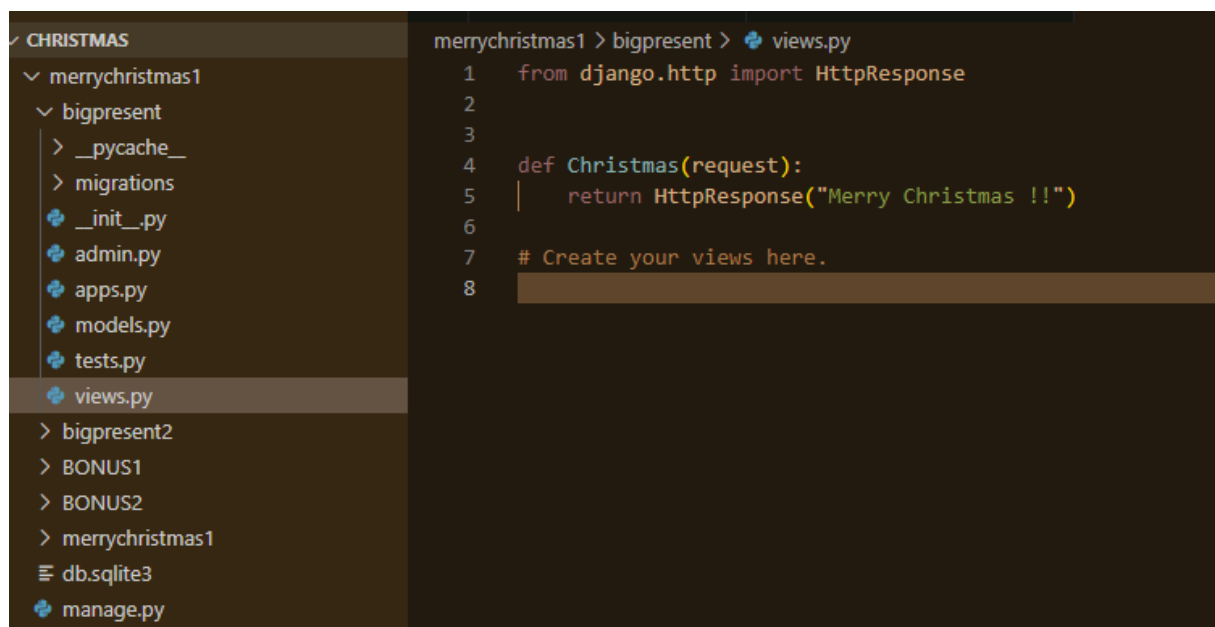
Basic

Initially my objective was to familiarize myself with django and nothing could be simpler than doing very basic things (as requested by the instructions!). As explained previously, I started a new django project by following the steps that I explained in the first part. Here is the architecture I got:



You can see in the image on the left a new project called "merrychristmas1" as well as two applications which are "bigpresent" and "bigpresent2". All this is grouped in a file called "christmas". The project contains two applications because I decided to group all my work from example 1 basic and example 2 into one project.

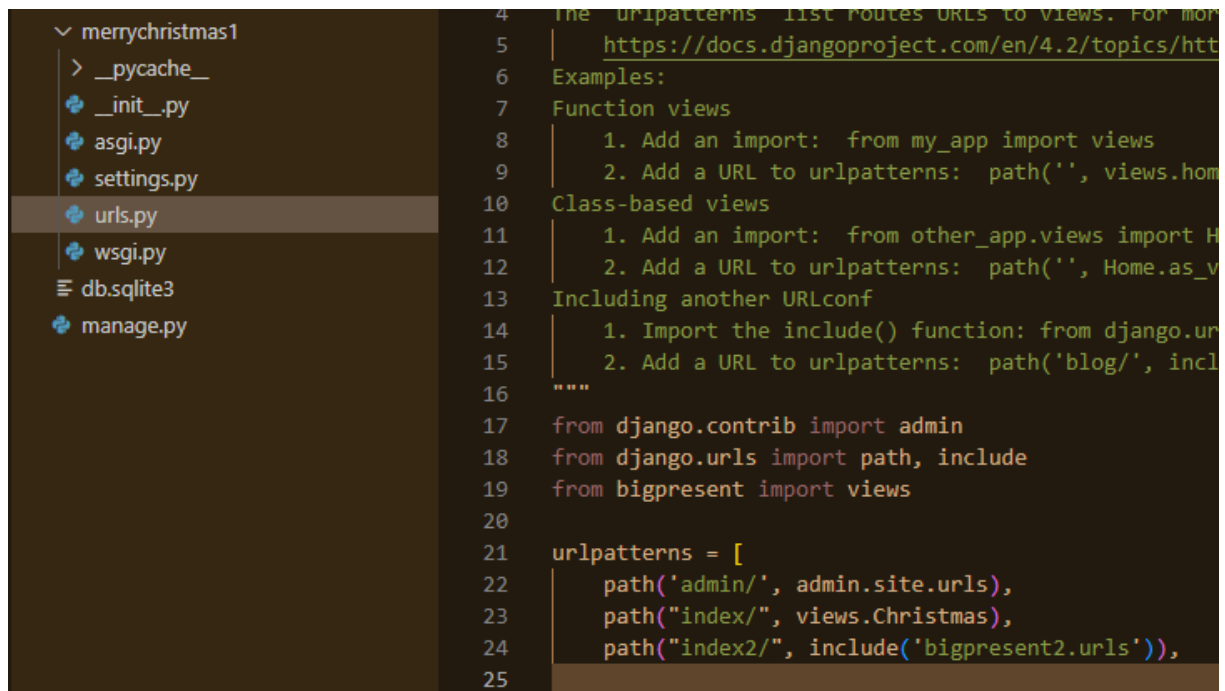
For the basic example we need to go to the "bigpresent" application. We can find the code which will give us our basic result in "views.py" which you can observe:



This function defines a django view function named "Christmas" with a request parameter. For django a view is a python function that takes a web request and returns a web response. Here it will return a web page with "Merry Christmas!!"

To better understand how things work with the web page, we need to look at the "urls.py" file which is located in the "merrychristmas1" project file:

This code allows you to correctly organize access to the different applications contained in the project.



The image shows a code editor with two panels. The left panel displays a file explorer for a project named 'merrychristmas1'. The files listed are: `__pycache__`, `__init__.py`, `asgi.py`, `settings.py`, `urls.py` (which is selected and highlighted), `wsgi.py`, `db.sqlite3`, and `manage.py`. The right panel shows the content of the `urls.py` file. It includes a comment about `urlpatterns` pointing to a Django documentation link, followed by examples for function and class-based views. The main code defines `urlpatterns` as a list of paths: `path('admin/', admin.site.urls)`, `path("index/", views.Christmas)`, and `path("index2/", include('bigpresent2.urls'))`. Imports for `admin`, `path`, `include`, and `views` are also shown.

For the first basic example we can observe it on the following web page: <http://127.0.0.1:8000/index/>

And the final result gives us this:

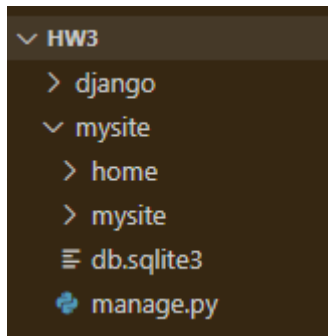


With this first approach to django I was able to display a message directly on a web page without using html code but simply with a character string included in a django view function.

More funny

After having understood the basic and simple elements of django and its architecture I decided to launch into a different code which this time includes an html code which will allow me (unlike earlier) to display things on my web page.

For the next two parts of example 1 we need to go to a different django project. My next two works are in a different project because I did not want to mix what was asked for in the instructions and what I have to carry out to quote.



Here is my architecture: a project called "mysite" and an application all grouped together in an "HW3" file.

There is nothing that changes in the architecture of the project compared to the one before. Now in the application there is a subfolder called "templates" which contains the HTML files that we will now study:

```
1. <html>
2.   <body style="color: rgb(231, 27, 180); font-size: 240px; background-image:
   url('https://img.freepik.com/photos-gratuite/fond-decoratif-detail-paillettes_23-
   2148210052.jpg?size=626&ext=jpg'); background-size: cover;">
3.     Hello world !!! :)
4.   </body>
5. </html>
```

In this first html code I wanted to write "Hello world!!!:)" on my internet page but that's not all, I enlarged the writing and put the text in pink color. In addition I placed an image in the background (this is what corresponds to the url in the code) which is placed to fill the entire background of the internet page. You can find this page directly on the following link: <http://127.0.0.1:8000/>



Christmas theme

For this third part of example 1 I decided to improve my previous html code by using animation. For this I used CSS and JS directly in the HTML code. For this new example I kept the previous project, I just changed the html code. The link remains the same.

Here is the HTML code I'm talking about:

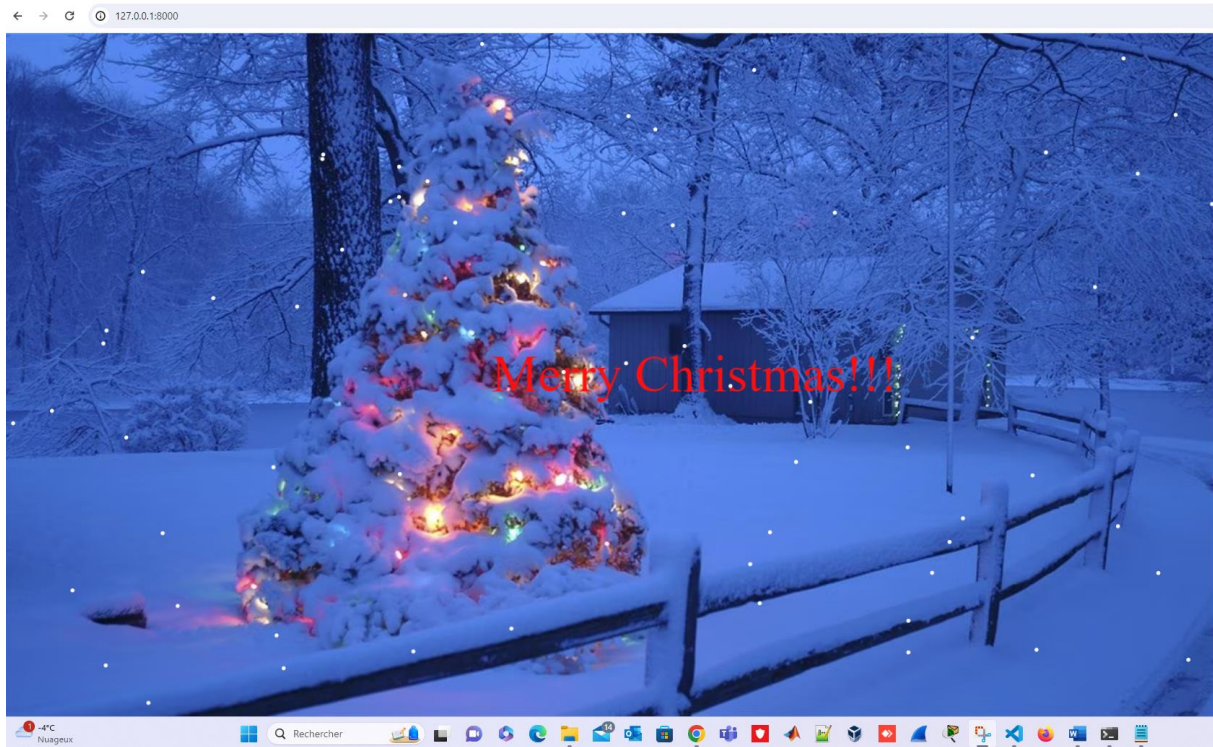
```
1. <html lang="en"> <!-- Opening tag for HTML, lang attribute set to "en" (English) -->
2.
3. <head>
4.   <!-- Head section containing metadata and links to external resources -->
5.   <meta charset="UTF-8"> <!-- Defines the character encoding -->
6.   <meta name="viewport" content="width=device-width, initial-scale=1.0"> <!-- Sets
   initial view and scale for mobile devices -->
7.   <title>Animated Greetings</title> <!-- Defines the page title -->
8.
9.   <!-- Embedded CSS styles -->
10.  <style>
11.    body {
12.      margin: 0;
13.      overflow: hidden;
14.      height: 100vh;
15.      display: flex;
16.      justify-content: center;
17.      align-items: center;
18.      background-image: url('https://img.radio-
        canada.ca/2015/12/10/1250x703/151210_em28y_hiver-fetes-noel-neige-
        froid_sn1250.jpg');
19.      background-size: contain;
20.      background-repeat: no-repeat;
21.      animation: colorChange 10s infinite, scaleText 3s infinite alternate;
22.    }
23.
24.    p {
25.      color: rgb(231, 27, 180);
26.      font-size: 60px;
27.      text-align: center;
28.      animation: colorChange 10s infinite, scaleText 3s infinite alternate;
29.    }
30.
31.    @keyframes colorChange {
32.      0% {
33.        color: rgb(250, 248, 249);
34.      }
35.      25% {
36.        color: #ff0000; /* Red */
37.      }
38.      50% {
39.        color: #228B22; /* Dark Green (forest green) */
40.      }
41.      75% {
42.        color: #191970; /* Midnight Blue */
43.      }
44.      100% {
45.        color: #2F4F4F; /* Dark Slate Gray */
46.      }
47.    }
48.
49.    @keyframes scaleText {
50.      from {
51.        transform: scale(1);
52.      }
```

```

53.     to {
54.         transform: scale(1.1);
55.     }
56. }
57.
58. /* Falling snow effect */
59. .snowflake {
60.     position: absolute;
61.     width: 5px;
62.     height: 5px;
63.     background-color: #ffffff;
64.     border-radius: 50%;
65.     animation: fall 5s linear infinite;
66. }
67.
68. @keyframes fall {
69.     0% {
70.         transform: translateY(-100vh);
71.     }
72.     100% {
73.         transform: translateY(100vh);
74.     }
75. }
76. </style>
77. </head>
78.
79. <body>
80.     <!-- Body section of the page -->
81.     <p>Merry Christmas!!!</p> <!-- Paragraph displaying the message "Merry
      Christmas!!!" -->
82.
83.     <!-- Adding snowflakes with JavaScript -->
84.     <script>
85.         // Generate 200 snowflakes
86.         for (let i = 0; i < 200; i++) {
87.             createSnowflake();
88.         }
89.
90.         function createSnowflake() {
91.             const snowflake = document.createElement('div');
92.             snowflake.className = 'snowflake';
93.             document.body.appendChild(snowflake);
94.
95.             // Random position at the top of the image
96.             const top = Math.random() * -100; // Starts above the image
97.             const left = Math.random() * window.innerWidth;
98.             const animationDuration = Math.random() * 5 + 3; // Animation duration between
      3 and 8 seconds
99.
100.             snowflake.style.top = `${top}px`;
101.             snowflake.style.left = `${left}px`;
102.             snowflake.style.animationDuration = `${animationDuration}s`;
103.         }
104.     </script>
105. </body>
106.
107. </html>

```

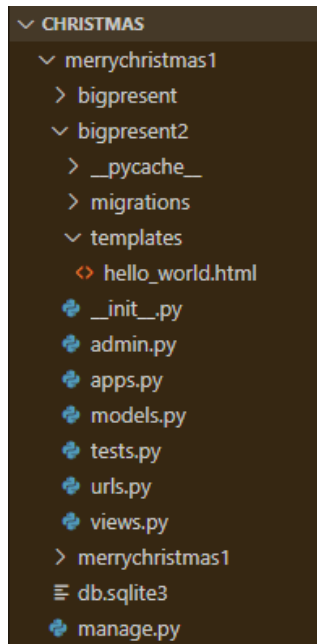

The final rendering is interactive, we can observe snow falling from the top of the image, the text which takes on different colors as it moves :



Example 2

In this second part, I recreated a web page that already exists in my own django project.

Return an HTML page

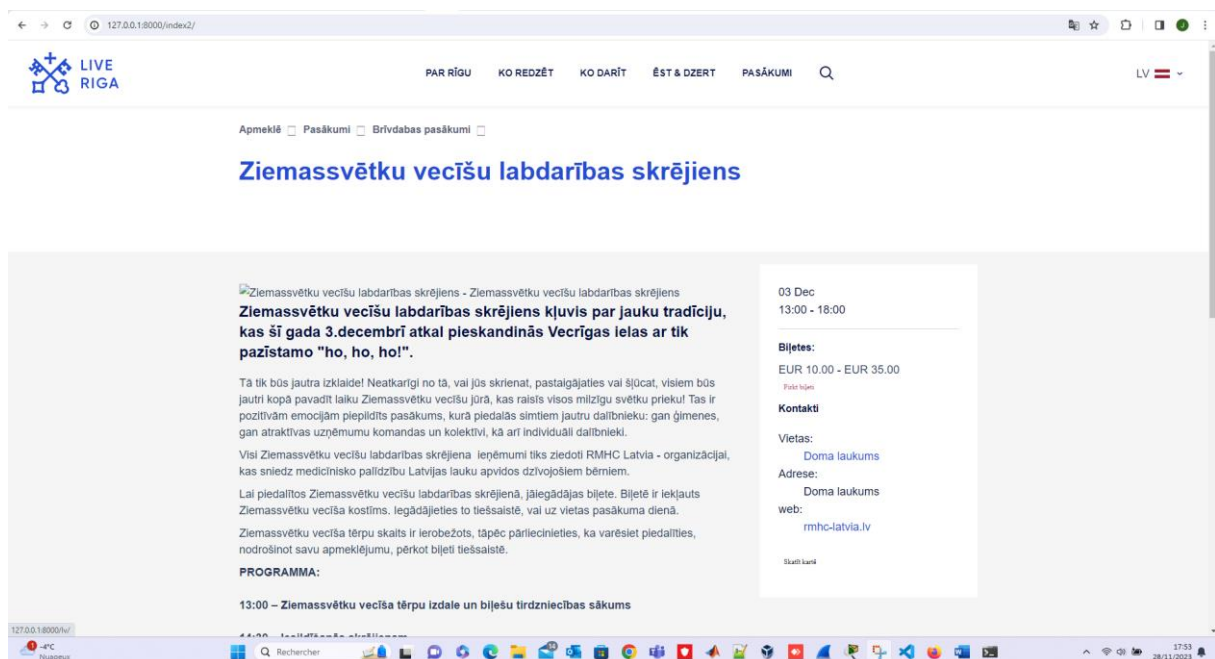


To do this you have to go back to the very first django project that I spoke to you about which contains the display "Meery christmas!" without html code. We now find ourselves in the "bigpresent2" application.

On the left you can observe the structure. Everything will happen in the "hello_world.html" file which is in the templates. In it is the html code of the (existing) web page that I am trying to reproduce.

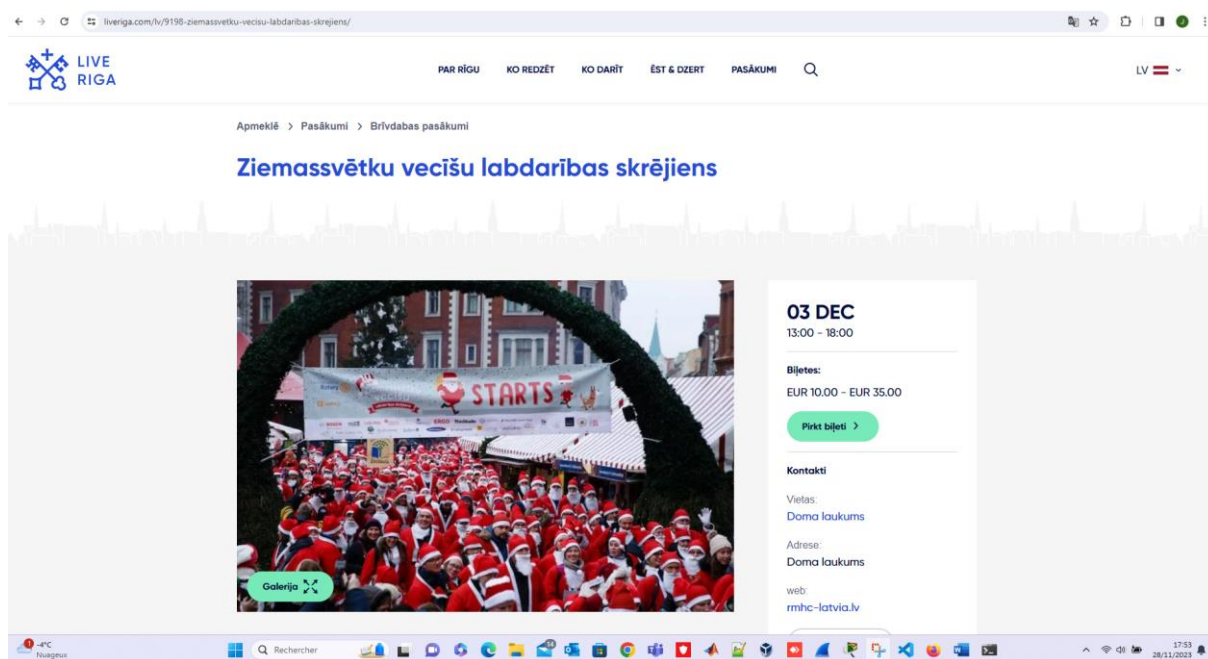
Once the project is compiled you can find this reproduction of the web page directly on the link: <http://127.0.0.1:8000/index2/>

The result that appears on the reproduction of the web page is the following: all the text elements are present as desired.



Here is the link to the original page: <https://www.liveriga.com/lv/9198-ziemassvetku-vecisu-labdaribas-skrejiens/>

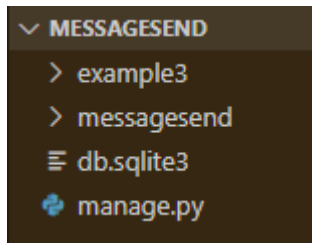
We can notice differences which are the presence of the images. In my html code of the page I was not able to understand and put the images as shown on this page.



Example 3

This example 3 was really my favorite part of this homework. We will first develop messaging between two people via two web pages then secondly we will create Santa Claus' messaging.

Classic messaging



To create this messaging I first created a new Django project by following the steps previously explained. The overall project is called "messagesend" and it contains an application named "example3". You can observe the architecture on the left.

Inside the "example3" application it was once again necessary to create an additional "templates" file which will contain the html codes of the two web pages: the message sending page and the receiving page.

Here are the commented codes of the different codes that I developed:

- « get_message.html »
- ```
<html lang="en"> <!-- Opening tag for HTML, lang attribute set to "en" (English) -->
•
• <head>
• <!-- Head section containing metadata -->
• <meta charset="UTF-8"> <!-- Defines the character encoding as UTF-8 -->
• <title>Message Board</title> <!-- Sets the title of the HTML document -->
•
• <!-- Internal CSS styles -->
• <style>
• body {
• margin: 0; /* Removes default margin */
• padding: 0; /* Removes default padding */
• overflow: hidden; /* Hides overflowing content */
• background:
• url('https://i.pinimg.com/originals/b4/a0/00/b4a000cdd7e39ca3dd3d2c6bb09872d0.jpg')
• no-repeat fixed center center; /* Sets a background image and fixes its position */
• background-size: cover; /* Ensures the background image covers the
• entire viewport */
• }
•
• .center-content {
• position: absolute; /* Absolute positioning */
• top: 50%; /* Positions the element at 50% from the top */
• left: 50%; /* Positions the element at 50% from the left */
• transform: translate(-50%, -50%); /* Centers the element by translating
• it 50% from both horizontal and vertical directions */
• text-align: center; /* Centers text within the container */
• color: rgb(255, 0, 128); /* Sets text color to flashy pink */
• }
•
• .message-box {
```



```

• background-color: rgb(202, 201, 201); /* Sets background color for
message boxes */
• padding: 10px; /* Adds padding inside message boxes */
• margin: 10px; /* Adds margin around message boxes */
• border-radius: 5px; /* Adds rounded corners to message boxes */
• }
•
• .message-info {
• color: rgb(0, 0, 0); /* Sets text color for message information */
• margin: 5px 0; /* Adds margin above and below message information */
• }
• </style>
• </head>
•
• <body>
• <!-- Body section of the page -->
• <div class="center-content">
• <!-- Centered content -->
• <h1 style="color: rgb(250, 0, 125);">Messages for {{ recipient_name }}</h1>
• <!-- Heading displaying recipient's name -->
•
• <!-- Loop through messages using template tags -->
• {% for message in messages %}
• <div class="message-box">
• <!-- Container for each message -->
• <p class="message-info">From: {{ message.sender_name }}</p>
• <!-- Display sender's name -->
• <p class="message-info">Message: {{ message.content }}</p>
• <!-- Display message content -->
• <p class="message-info">Sent at: {{ message.timestamp }}</p>
• <!-- Display message timestamp -->
• </div>
• {% endfor %}
• <!-- End of the loop -->
• </div>
• </body>
•
• </html>

```

• « submit\_message.html »

```

• <html lang="en"> <!-- Opening tag for HTML, lang attribute set to "en" (English) -->
•
• <head>
• <!-- Head section containing metadata -->
• <meta charset="UTF-8"> <!-- Defines the character encoding as UTF-8 -->
• <title>Submit Message</title> <!-- Sets the title of the HTML document -->
•
• <!-- Internal CSS styles -->
• <style>
• body {
• margin: 0; /* Removes default margin */
• padding: 0; /* Removes default padding */
• overflow: hidden; /* Hides overflowing content */
• background:
• url('https://i.pinimg.com/originals/b4/a0/00/b4a000cdd7e39ca3dd3d2c6bb09872d0.jpg')
• no-repeat fixed center center; /* Sets a background image and fixes its position */
• background-size: cover; /* Ensures the background image covers the
entire viewport */
• }
• </style>

```



```

•
• .center-content {
• position: absolute; /* Absolute positioning */
• top: 50%; /* Positions the element at 50% from the top */
• left: 50%; /* Positions the element at 50% from the left */
• transform: translate(-50%, -50%); /* Centers the element by translating
it 50% from both horizontal and vertical directions */
• text-align: center; /* Centers text within the container */
• color: black; /* Sets text color to black */
• }
•
• .send-button {
• background-color: hotpink; /* Sets background color to flashy pink */
• color: black; /* Sets text color to black */
• padding: 10px 20px; /* Adds padding to the button */
• font-size: 16px; /* Sets the font size */
• border: none; /* Removes button border */
• cursor: pointer; /* Changes cursor to pointer on hover */
• }
• </style>
• </head>
•
• <body>
• <!-- Body section of the page -->
• <div class="center-content">
• <!-- Centered content -->
• <h2>Submit a Message</h2>
• <!-- Heading indicating the purpose of the form -->
•
• <!-- Form for submitting a message -->
• <form method="post">
• {% csrf_token %} <!-- CSRF token for security -->
• {{ form.as_p }} <!-- Renders form fields as paragraphs -->
• <button class="send-button" type="submit">Submit</button>
• <!-- Submit button with styling -->
• </form>
• </div>
• </body>
•
• </html>

```

• « models.py »

This model defines the structure and behavior of ‘Message’ object in a Django application. It will be used to create, query, and manage messages in the associated database.

```

1. # Django model definition for a 'Message'
2. class Message(models.Model):
3. # CharField for the sender's name with a maximum length of 100 characters
4. sender_name = models.CharField(max_length=100)
5.
6. # CharField for the recipient's name with a maximum length of 100 characters
7. recipient_name = models.CharField(max_length=100)
8.
9. # TextField for the content of the message (allows longer text)
10. content = models.TextField()
11.
12. # DateTimeField to store the timestamp when the message is created (auto-generated
13. on creation)
14. timestamp = models.DateTimeField(auto_now_add=True)
15.

```

```

16. # String representation of the Message object
17. def __str__(self):
18. # Returns a formatted string representing the message, including sender, recipient,
19. and timestamp
20. return f'{self.sender_name} to {self.recipient_name} - {self.timestamp}'

```

- « urls.py »

```

1. from django.urls import path
2. from .views import submit_message, get_message
3. from . import views
4.
5. urlpatterns = [
6. path('submit/', submit_message, name='submit_message'),
7. path('get/<str:recipient_name>', get_message, name='get_message'),
8.
9.]

```

- « views.py »

```

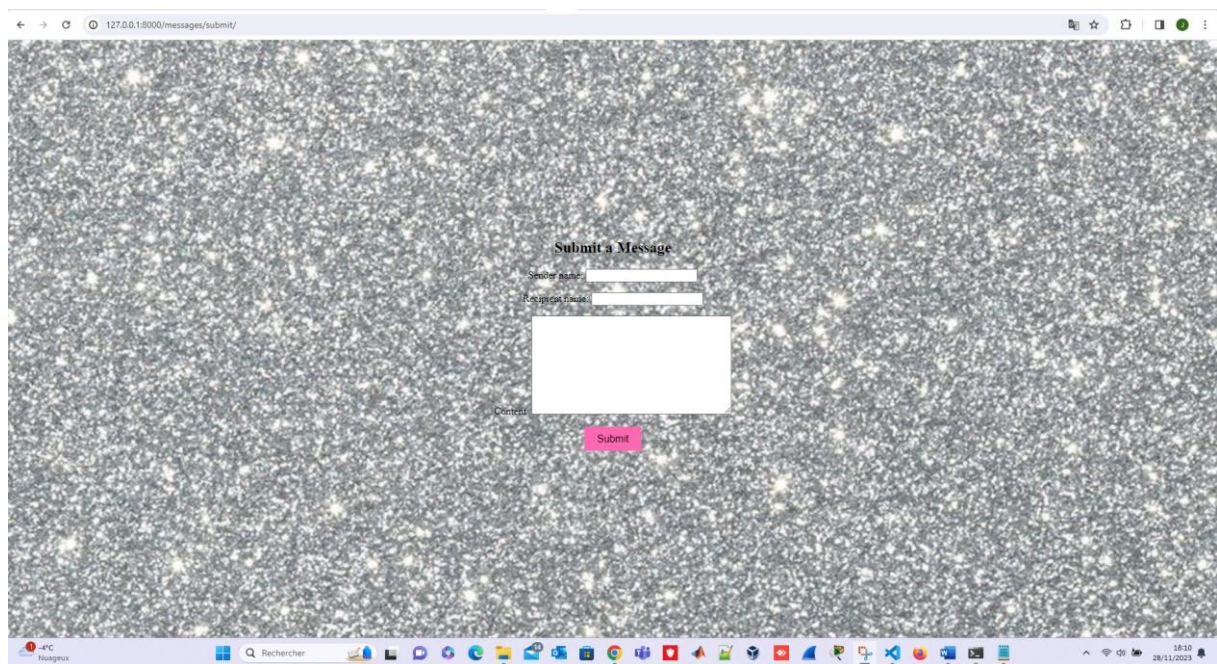
1. def submit_message(request):
2. # Check if the request method is POST
3. if request.method == 'POST':
4. # Create a MessageForm instance with the data from the request
5. form = MessageForm(request.POST)
6.
7. # Check if the form is valid
8. if form.is_valid():
9. # Save the form data to the database
10. form.save()
11.
12. # Redirect to the 'submit_message' view (to avoid form resubmission)
13. return redirect('submit_message')
14. else:
15. # If the request method is not POST, create an empty MessageForm instance
16. form = MessageForm()
17.
18. # Render the 'submit_message.html' template with the form
19. return render(request, 'submit_message.html', {'form': form})
20.
21. def get_message(request, recipient_name):
22. # Query the database for messages with the specified recipient_name, ordered by
23. timestamp (latest first), and limit to 20 messages
24. messages = Message.objects.filter(recipient_name=recipient_name).order_by('-
25. timestamp')[:20]
26.
27. # Render the 'get_messages.html' template with the messages and recipient_name
28. return render(request, 'get_messages.html', {'messages': messages,
29. 'recipient_name': recipient_name})

```

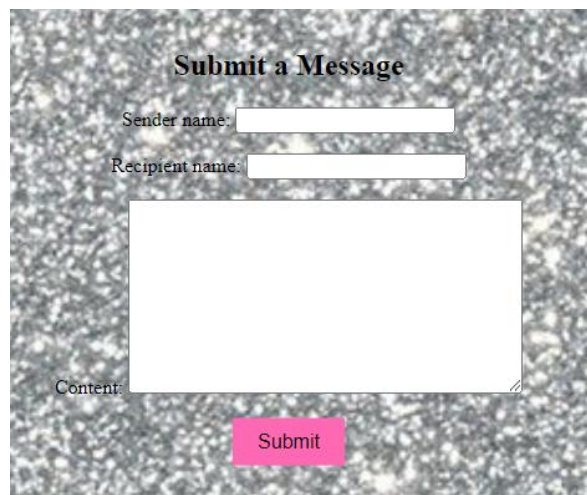
**submit\_message** : This view handles both GET and POST requests. If the request method is POST, it validates the form data, saves it to a databases usins ‘form.save()’ and redirects to the same views to prevent form resubmission. If the request method is not POST, it create an empty form to display on the page. The ‘MessageForm’ used in this view is assumed to be a Django form class for the ‘Message’ model.

**get\_message** : This view tales a ‘recipient\_name’ parameter from the URL. It queries the database for message with the specified ‘recipient\_name’, orders them by timestamp in descending order, and limits the result to the latest 20 messages. The results are passed to the ‘get\_messages.html’ template for rendering.

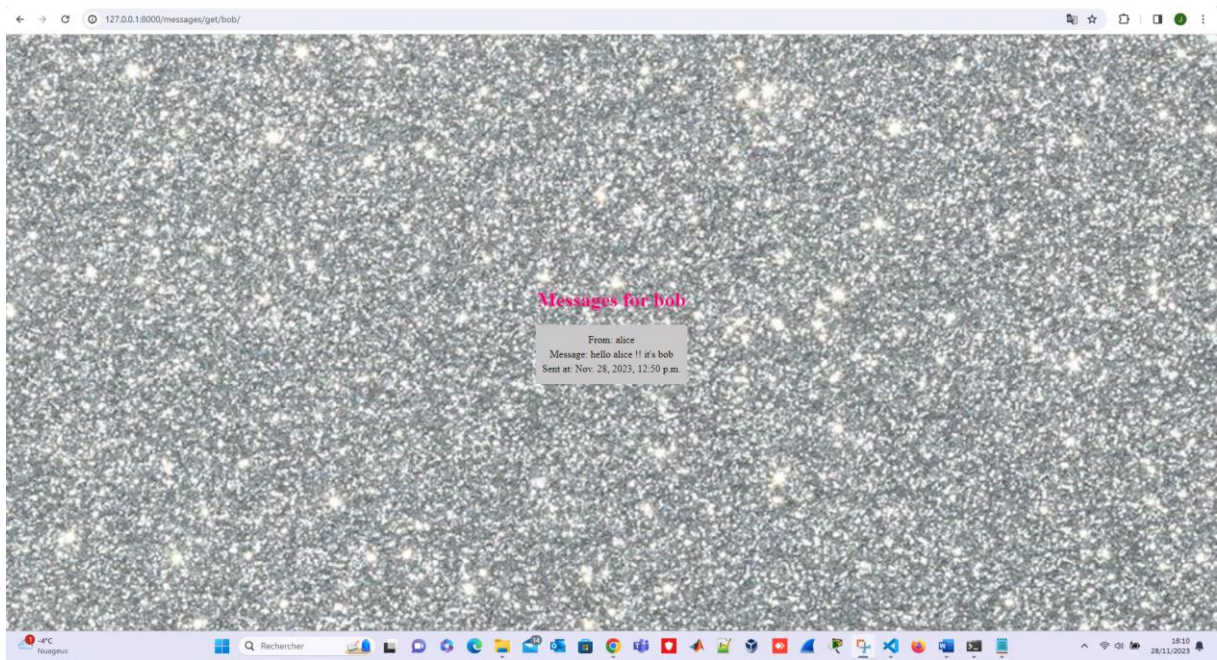
Here is the result of the two pages:



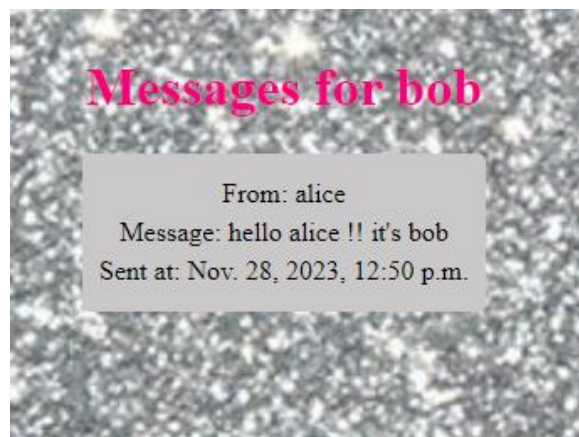
ZOOM





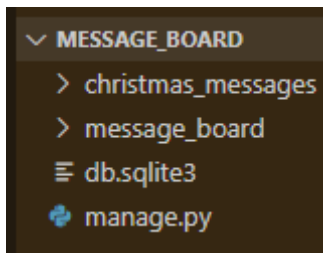


ZOOM



## Santa claus mail

In this part I decided to create Santa's messaging. For this I started with my knowledge acquired in the previous part of sending messages. I simply added things when sending the message such as knowing if the child has had caries over the years, an estimate of his level of kindness from 1 to 10 or even the 5 gifts he would like for Christmas ! In addition, it is not necessary to specify a recipient here since all the messages are for Santa Claus!!!! I also added for Santa Claus the possibility of deleting messages from his inbox on the principle that once he has made the children's gifts it is no longer necessary to keep their list...



As you can see I started a new django project for this new example. This one is called "message\_board" and contains an application called "christmas\_messages". As you can imagine, we will find the same file architecture as before with a templates folder and two html files for submission and reception.

To avoid repeating myself I decided to present to you only 3 codes compared to earlier which are the two html codes (I am very happy with my rendering) as well as the views which contains the two python functions (+1 for delete).

- “get\_messages.html”

```
<html lang="en"> <!-- Opening tag for HTML, lang attribute set to "en" (English) -->
<head>
 <!-- Head section containing metadata -->
 <meta charset="UTF-8"> <!-- Defines the character encoding as UTF-8 -->
 <meta name="viewport" content="width=device-width, initial-scale=1.0"> <!-- Sets the viewport for responsive design -->
 <title>Christmas Messages to Santa</title> <!-- Sets the title of the HTML document -->
 <!-- Internal CSS styles -->
 <style>
 body {
 background-color: #800020; /* Burgundy Red background color */
 color: rgb(12, 10, 10); /* Dark text color */
 font-family: Arial, sans-serif; /* Font family for the entire page */
 margin: 0; /* Removes default margin */
 padding: 0; /* Removes default padding */
 text-align: center; /* Aligns text to the center */
 }
 .messages-container {
 max-width: 800px; /* Sets the maximum width of the container */
 margin: 20px auto; /* Centers the container and adds margin */
 }
 .message-list {
 list-style-type: none; /* Removes default list styling */
 padding: 0; /* Removes default padding for lists */
 }
 .message-item {
```



```

 background-color: #FFFFFF; /* White background color for each message
 item */
 border: 1px solid #191970; /* Navy Blue border */
 border-radius: 10px; /* Adds rounded corners */
 margin: 10px; /* Adds margin around each message item */
 padding: 10px; /* Adds padding inside each message item */
 text-align: left; /* Aligns text to the left */
}

.delete-button {
 background-color: #800020; /* Burgundy Red background color for the
delete button */
 color: white; /* White text color for the delete button */
 padding: 5px 10px; /* Adds padding to the delete button */
 border: none; /* Removes button border */
 border-radius: 5px; /* Adds rounded corners to the delete button */
 cursor: pointer; /* Changes cursor to pointer on hover */
}
</style>
</head>
<body>
 <!-- Body section of the page -->
 <div class="messages-container">
 <!-- Container for displaying Christmas messages -->
 <h1>Latest Christmas Messages to Santa</h1>
 <!-- Heading indicating the purpose of the page -->

 <ul class="message-list">
 <!-- Unordered list for displaying messages -->
 {% for message in messages %}
 <!-- Loop through each Christmas message -->
 <li class="message-item">
 <!-- List item for each message -->
 <strong style="color: #191970;">{{ message.sender_name }} {{
message.last_name }} says:

 <!-- Display sender's name in bold with Navy Blue color -->
 Gift List: {{ message.gift_list }}

 <!-- Display gift list -->
 Kindness Level: {{ message.kindness_level }}

 <!-- Display kindness level -->
 Age: {{ message.age }}

 <!-- Display age -->
 {% if message.has_cavities %}Has Cavities{% else %}No Cavities{%
endif %}

 <!-- Display cavities status -->
 City: {{ message.city }}

 <!-- Display city -->
 Sent on: {{ message.timestamp }}

 <!-- Display timestamp -->
 <form method="post" action="{% url 'delete_message' message.id
%}">
 <!-- Form for deleting a message -->
 {% csrf_token %}
 <!-- CSRF token for security -->
 <button class="delete-button" type="submit">Delete</button>
 <!-- Delete button with styling -->
 </form>

 <!-- Horizontal line to separate messages -->
 {% endfor %}

```

- <!-- End of the loop -->
- </ul>
- </div>
- </body>
- </html>

- “submit\_message.html”

```

1. <html lang="en"> <!-- Opening tag for HTML, lang attribute set to "en" (English) -->
2.
3. <head>
4. <!-- Head section containing metadata -->
5. <meta charset="UTF-8"> <!-- Defines the character encoding as UTF-8 -->
6. <meta name="viewport" content="width=device-width, initial-scale=1.0"> <!-- Sets
 the viewport for responsive design -->
7. <title>Submit Your Christmas Message to Santa</title> <!-- Sets the title of the
 HTML document -->
8.
9. <!-- Internal CSS styles -->
10. <style>
11. body {
12. background-image:
13. url('https://i.ytimg.com/vi/U33p70iwd5E/maxresdefault.jpg'); /* Replace with the
 path to your background image */
14. background-size: cover; /* Ensures the background image covers the
 entire viewport */
15. background-position: center; /* Centers the background image */
16. margin: 0; /* Removes default margin */
17. display: flex; /* Uses flexbox for layout */
18. align-items: center; /* Centers content vertically */
19. justify-content: center; /* Centers content horizontally */
20. height: 100vh; /* Sets the height of the body to 100% of the viewport
 height */
21. }
22. .form-container {
23. background-color: #800020; /* Burgundy Red background color for the form
 container */
24. padding: 20px; /* Adds padding inside the form container */
25. border-radius: 10px; /* Adds rounded corners to the form container */
26. width: 400px; /* Sets the width of the form container */
27. text-align: center; /* Centers text inside the form container */
28. color: white; /* White text color for the form content */
29. }
30. label {
31. display: block; /* Displays labels as block elements to separate them */
32. margin-bottom: 10px; /* Adds space below each label */
33. }
34. input, textarea, select {
35. width: 100%; /* Makes form inputs take up 100% of the container width */
36. padding: 8px; /* Adds padding inside form inputs */
37. margin-bottom: 15px; /* Adds space below each form input */
38. box-sizing: border-box; /* Ensures padding and border are included in
 the total width/height */
39. }
40. button {
41. background-color: #228B22; /* Green background color for the submit
 button */

```

```

45. color: #800020; /* Burgundy Red text color for the submit button */
46. padding: 10px 20px; /* Adds padding to the submit button */
47. border: none; /* Removes button border */
48. border-radius: 5px; /* Adds rounded corners to the submit button */
49. cursor: pointer; /* Changes cursor to pointer on hover */
50. }
51. </style>
52. </head>
53.
54. <body>
55. <!-- Body section of the page -->
56. <div class="form-container">
57. <!-- Container for the Christmas message submission form -->
58. <h1>Submit Your Christmas Message to Santa</h1>
59. <!-- Heading indicating the purpose of the form -->
60.
61. <form method="post" action="{% url 'submit_message' %}">
62. <!-- Form for submitting Christmas messages -->
63. {% csrf_token %}
64. <!-- CSRF token for security -->
65.
66. <label for="sender_name">First Name:</label>
67. <!-- Label for the first name input -->
68. <input type="text" name="sender_name" required>

69. <!-- Text input for the first name -->
70.
71. <label for="last_name">Last Name:</label>
72. <!-- Label for the last name input -->
73. <input type="text" name="last_name" required>

74. <!-- Text input for the last name -->
75.
76. <label for="city">City of Residence:</label>
77. <!-- Label for the city input -->
78. <input type="text" name="city" required>

79. <!-- Text input for the city -->
80.
81. <label for="gift_list">Gift List (up to 5 gifts, separated by
commas):</label>
82. <!-- Label for the gift list textarea -->
83. <textarea name="gift_list" required></textarea>

84. <!-- Textarea for the gift list -->
85.
86. <label for="kindness_level">Kindness Level (1-10):</label>
87. <!-- Label for the kindness level input -->
88. <input type="number" name="kindness_level" min="1" max="10"
required>

89. <!-- Number input for the kindness level -->
90.
91. <label for="age">Your Age:</label>
92. <!-- Label for the age input -->
93. <input type="number" name="age" required>

94. <!-- Number input for the age -->
95.
96. <label for="has_cavities">Do you have cavities?</label>
97. <!-- Label for the cavities checkbox -->
98. <input type="checkbox" name="has_cavities">

99. <!-- Checkbox for indicating cavities status -->
100.
101. <button type="submit">Submit</button>
102. <!-- Submit button for submitting the Christmas message -->
103. </form>
104. </div>
105. </body>
106.
107. </html>

```

- “views.py”

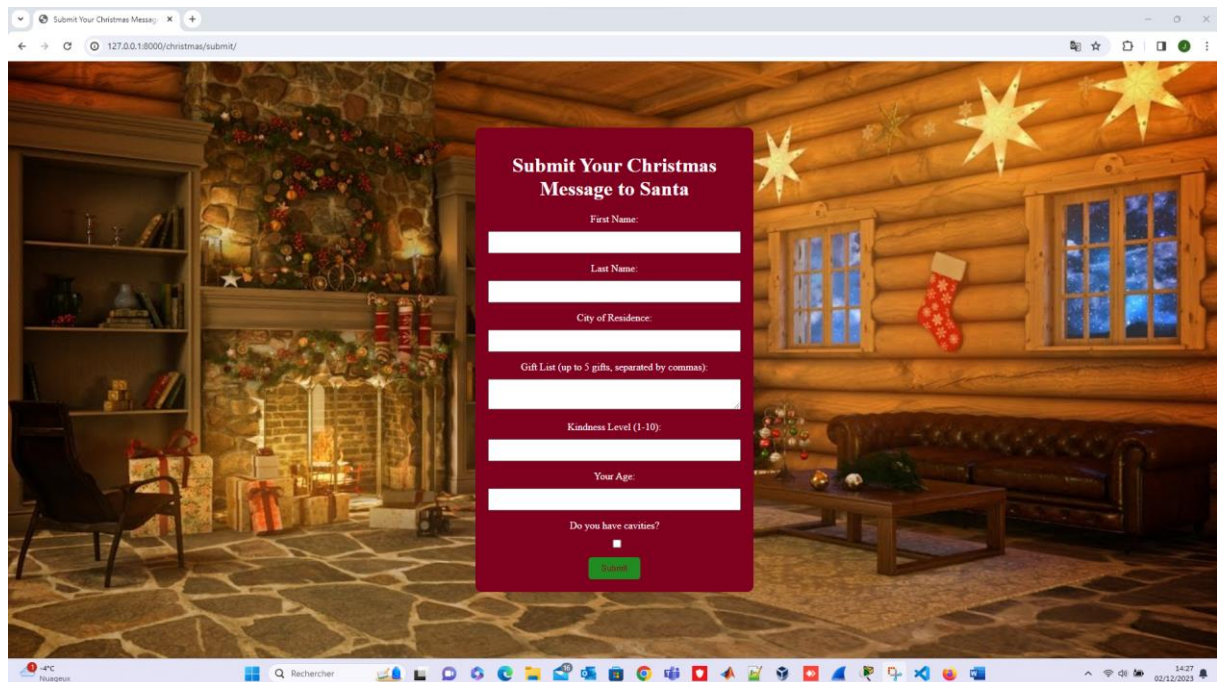
```

1. # View for submitting a Christmas message
2. def submit_message(request):
3. # Check if the request method is POST
4. if request.method == 'POST':
5. # Retrieve data from the POST request
6. sender_name = request.POST.get('sender_name')
7. last_name = request.POST.get('last_name') # Added last_name
8. city = request.POST.get('city') # Added city
9. gift_list = request.POST.get('gift_list')
10. kindness_level = request.POST.get('kindness_level')
11. age = request.POST.get('age')
12.
13. # Handle the 'has_cavities' field correctly
14. has_cavities = request.POST.get('has_cavities', False)
15. has_cavities = True if has_cavities == 'on' else False
16.
17. # Create a new ChristmasMessage object with the submitted data
18. ChristmasMessage.objects.create(
19. sender_name=sender_name,
20. last_name=last_name, # Added last_name
21. city=city, # Added city
22. gift_list=gift_list,
23. kindness_level=kindness_level,
24. age=age,
25. has_cavities=has_cavities
26.)
27.
28. # Redirect the user to the starting page after submitting the message
29. return redirect('submit_message')
30.
31. # Render the submit_message.html template for GET requests
32. return render(request, 'submit_message.html')
33.
34. # View for retrieving all Christmas messages
35. def get_messages(request):
36. # Retrieve all Christmas messages, ordered by timestamp in descending order
37. messages = ChristmasMessage.objects.order_by('-timestamp')
38.
39. # Render the get_messages.html template with the messages
40. return render(request, 'get_messages.html', {'messages': messages})
41.
42. # View for deleting a Christmas message
43. def delete_message(request, message_id):
44. # Retrieve the ChristmasMessage object with the given message_id
45. message = ChristmasMessage.objects.get(pk=message_id)
46.
47. # Delete the message
48. message.delete()
49.
50. # Redirect the user to the get_messages view after deleting the message
51. return redirect('get_messages')

```

These Django views are designed to handle the submission, retrieval and deletion of Christmas messages. The submit\_messages view handles both GET and POST requests. For POST requests, it extracts form data, creates a new ChristmasMessage object and redirects the user to the submission page. For GET requests, it renders the submit\_message.html template. The get\_messages view retrieves all Christmas messages from the database, orders them by timestamp in descending order and renders the get\_messages.html template with the messages. The delete\_message view deletes a Christmas message based on the provided message\_id and redirects the user to the get\_messages view.

Here are the final renderings of the two pages:



**Submit Your Christmas Message to Santa**

First Name:

Last Name:

City of Residence:

Gift List (up to 5 gifts, separated by commas):

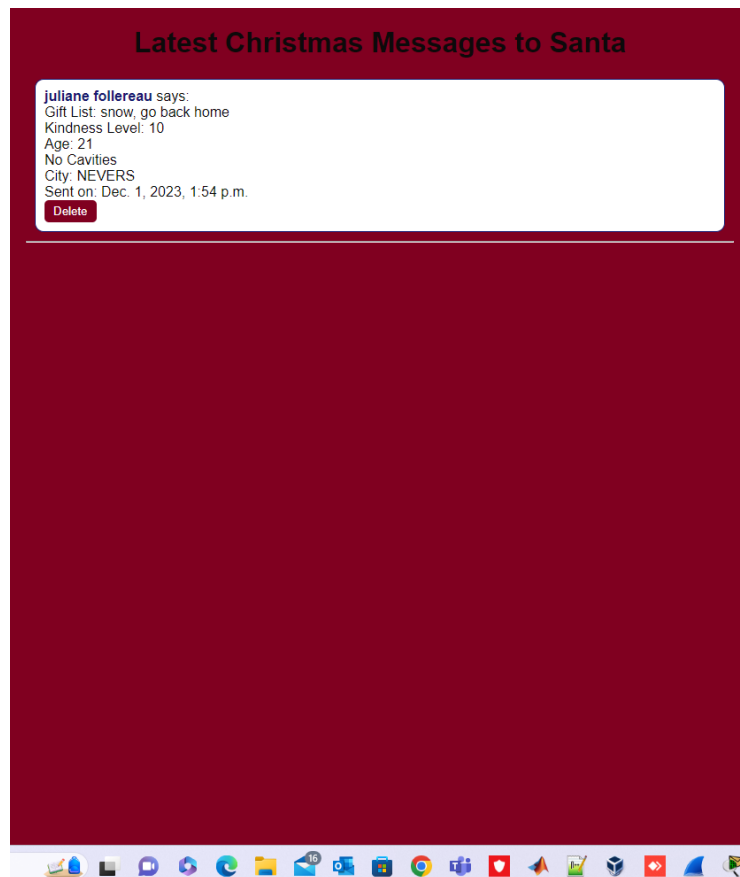
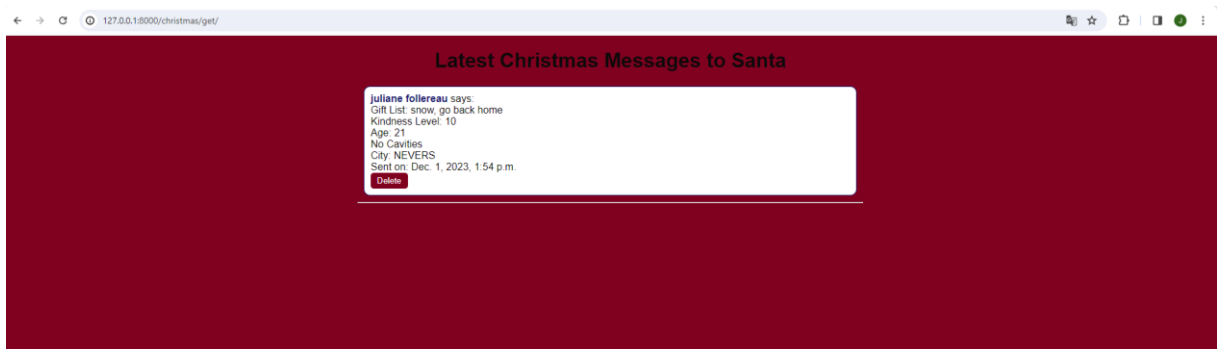
Kindness Level (1-10):

Your Age:

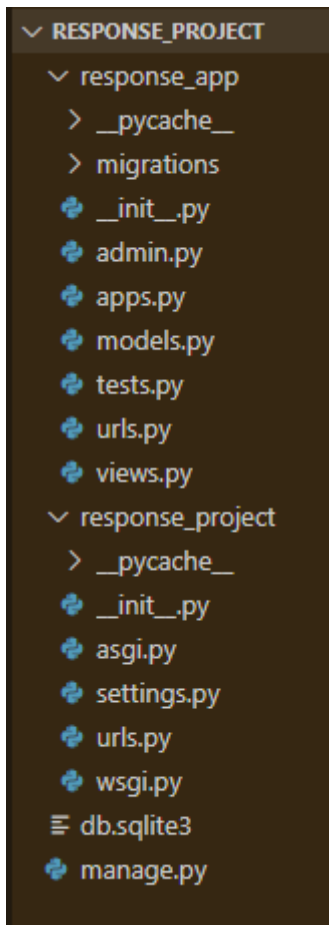
Do you have cavities?

☐





## Example 4



For this last example I once again created a new Django project! You can see how it is constructed on the left with its tree structure. I decided to create 14 URL links for each method / class that I will test in this new Django project. First I will show you the two most important codes of this project then we will go into more detail about each method with the explanation of what it does previously and especially what it returns when you enter it. URL that matches it!

Response\_app > views.py :

```
1. from django.shortcuts import render
2. from django.views import View
3. from django.http import HttpResponseRedirect, HttpResponsePermanentRedirect, \
4. HttpResponseNotModified, HttpResponseBadRequest, HttpResponseForbidden, \
5. HttpResponseNotAllowed, HttpResponseGone, HttpResponseServerError, \
6. HttpResponseNotFound, JsonResponse, StreamingHttpResponse, FileResponse
7.
8. class MainResponseView(View):
9.
10. def get(self, request, *args, **kwargs):
11. return HttpResponseRedirect("Hello, this is the main HttpResponseRedirect")
12.
13. class RedirectedView(View):
14. def get(self, request, *args, **kwargs):
15.
16. return HttpResponseRedirect('/example/')
17.
18. class PermanentRedirectedView(View):
19. def get(self, request, *args, **kwargs):
20.
21. return HttpResponseRedirectPermanentRedirect('/example/')
22.
23. class NotModifiedView(View):
24. def get(self, request, *args, **kwargs):
25.
26. return HttpResponseRedirectNotModified()
27.
28. class BadRequestView(View):
29.
```

```

30. def get(self, request, *args, **kwargs):
31. return HttpResponseBadRequest("Bad Request")
32.
33. class ForbiddenView(View):
34.
35. def get(self, request, *args, **kwargs):
36. return HttpResponseForbidden("Forbidden")
37.
38. class MethodNotAllowedView(View):
39.
40. def get(self, request, *args, **kwargs):
41. return HttpResponseNotAllowed(["GET", "POST"], "Not Allowed")
42.
43. class GoneView(View):
44.
45. def get(self, request, *args, **kwargs):
46. return HttpResponseGone("Gone")
47.
48. class ServerErrorView(View):
49.
50.
51. def get(self, request, *args, **kwargs):
52. return HttpResponseServerError("Server Error")
53.
54. class NotFoundView(View):
55.
56. def get(self, request, *args, **kwargs):
57. return HttpResponseNotFound("Not Found")
58.
59. class JsonResponseView(View):
60.
61. def get(self, request, *args, **kwargs):
62. data = {'key': 'value'}
63. return JsonResponse(data)
64.
65. class StreamingResponseView(View):
66.
67. def get(self, request, *args, **kwargs):
68. content = "This is a streaming response."
69. return StreamingHttpResponse(content, content_type="text/plain")
70.
71. class FileResponseView(View):
72.
73. def get(self, request, *args, **kwargs):
74. file_path = 'C:/Users/julia/OneDrive/Documents/A4/RTU/Telecommunications
Software/test.txt'
75. return FileResponse(open(file_path, 'rb'), content_type="text/plain")
76.

```

Response\_project > urls.py :

```

1. from django.urls import path
2. from response_app.views import (
3. MainResponseView, RedirectedView, PermanentRedirectedView, NotModifiedView,
4. BadRequestView, ForbiddenView, MethodNotAllowedView, GoneView, ServerErrorView,
5. NotFoundView, JsonResponseView, StreamingResponseView, FileResponseView
6.)
7.
8. urlpatterns = [
9. path('example/', MainResponseView.as_view(), name='main_response'),
10. path('example/redirected/', RedirectedView.as_view(), name='redirected_response'),
11. path('example/permanent-redirected/', PermanentRedirectedView.as_view(),
name='permanent_redirected_response'),
12. path('example/not-modified/', NotModifiedView.as_view(), name='not_modified_response'),
13. path('example/bad-request/', BadRequestView.as_view(), name='bad_request_response'),
14. path('example/forbidden/', ForbiddenView.as_view(), name='forbidden_response'),
15. path('example/not-allowed/', MethodNotAllowedView.as_view(),
name='not_allowed_response'),

```

```

16. path('example/gone/', GoneView.as_view(), name='gone_response'),
17. path('example/server-error/', ServerErrorView.as_view(), name='server_error_response'),
18. path('example/not-found/', NotFoundView.as_view(), name='not_found_response'),
19. path('example/json-response/', JsonResponseView.as_view(), name='json_response'),
20. path('example/streaming-response/', StreamingResponseView.as_view(),
name='streaming_response'),
21. path('example/file-response/', FileResponseView.as_view(), name='file_response'),
22.]
23.

```

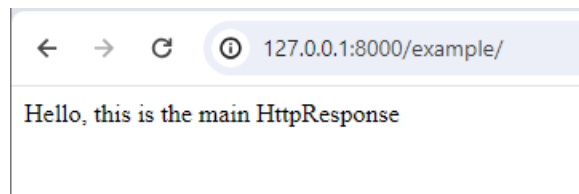
- HttpResponse

```

1. class MainResponseView(View):
2. def get(self, request, *args, **kwargs):
3. return HttpResponse("Hello, this is the main HttpResponse")
4.

```

This is the most basic method. When someone accesses this page, it simply returns a message here which is: "Hello, this is the main HttpResponse". Here is the rendering:



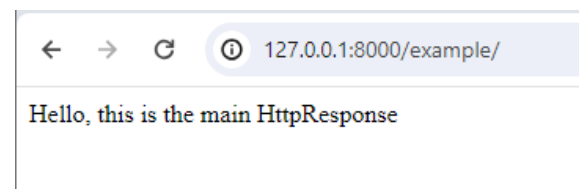
- HttpResponseRedirect

```

1. class RedirectedView(View):
2. def get(self, request, *args, **kwargs):
3.
4. return HttpResponseRedirect('/example/')
5.

```

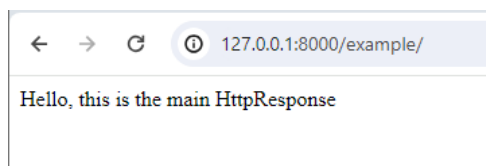
This method represents a temporary redirection to another page. When we access this page, it automatically redirects to the main view that we created previously after a few seconds. Here is the rendering:



- HttpResponseRedirect

```
1. class HttpResponseRedirect(View):
2. def get(self, request, *args, **kwargs):
3.
4. return HttpResponseRedirect('/example/')
5.
```

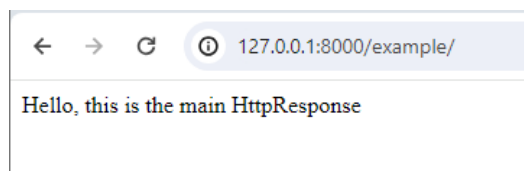
This method represents a permanent redirection to another page. When this page is accessed, it automatically redirects to the main view permanently. Exactly like the previous method but this time permanently. Here is the result :



- HttpResponseNotModified

```
1. class HttpResponseNotModifiedView(View):
2. def get(self, request, *args, **kwargs):
3.
4. return HttpResponseNotModified()
5.
```

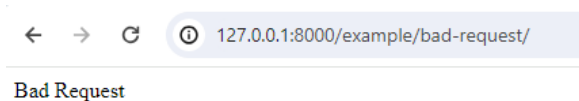
This method represents a response indicating that the resource has not been modified. It is generally used to save server bandwidth by avoiding returning the same resource if it has not changed since the last request. Here it will return the same page as the previous one since it has not been modified! The result is the following:



- HttpResponseBadRequest

```
1. class BadRequestView(View):
2.
3. def get(self, request, *args, **kwargs):
4. return HttpResponseBadRequest("Bad Request")
5.
```

This method represents a response indicating an incorrect request (bad request). So here we will have a display of a page with the message "Bad Request". Here is the result :

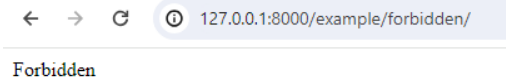




- `HttpResponseForbidden`

```
1. class ForbiddenView(View):
2.
3. def get(self, request, *args, **kwargs):
4. return HttpResponseForbidden("Forbidden")
5.
```

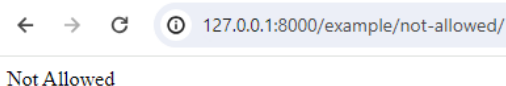
This method represents a response indicating that access to the resource is forbidden (forbidden). So we display a page with the message "Forbidden". Here is the result :



- `HttpResponseNotAllowed`

```
1. class MethodNotAllowedView(View):
2.
3. def get(self, request, *args, **kwargs):
4. return HttpResponseNotAllowed(["GET", "POST"], "Not Allowed")
5.
```

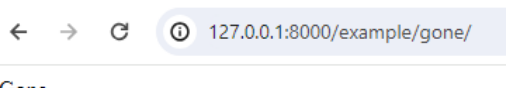
This method represents a response indicating that the HTTP method used is not allowed for this resource. We display a page with the message "Not Allowed" for HTTP methods other than GET or POST, which gives this:



- `HttpResponseGone`

```
1. class GoneView(View):
2.
3. def get(self, request, *args, **kwargs):
4. return HttpResponseGone("Gone")
5.
```

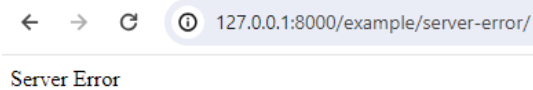
This method represents a response indicating that the requested resource is no longer available and has been removed. We display a page with the message "Gone":



- HttpResponseRedirect

```
1. class ServerErrorView(View):
2.
3.
4. def get(self, request, *args, **kwargs):
5. return HttpResponseRedirect("Server Error")
6.
```

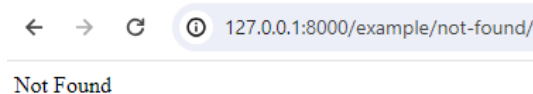
This method represents a response indicating an internal server error. We display a page with the message “Server Error”:



- HttpResponseRedirect

```
1. class NotFoundView(View):
2.
3. def get(self, request, *args, **kwargs):
4. return HttpResponseRedirect("Not Found")
5.
```

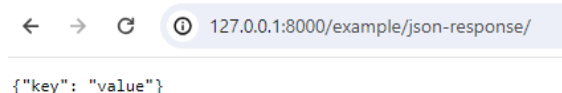
This method represents a response indicating that the requested resource was not found on the server. We display a page with the message "Not Found":



- JsonResponse

```
1. class JsonResponseView(View):
2.
3. def get(self, request, *args, **kwargs):
4. data = {'key': 'value'}
5. return JsonResponse(data)
6.
```

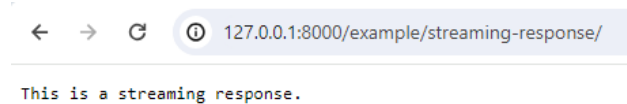
This method represents a response in JSON format. Here we display a page with the specified JSON data. like that :



- StreamingHttpResponse

```
1. class StreamingResponseView(View):
2.
3. def get(self, request, *args, **kwargs):
4. content = "This is a streaming response."
5. return StreamingHttpResponse(content, content_type="text/plain")
6.
```

This method represents a response with streaming content. We therefore display a page with streaming content:



- FileResponse

```
1. class FileResponseView(View):
2.
3. def get(self, request, *args, **kwargs):
4. file_path = 'C:/Users/julia/OneDrive/Documents/A4/RTU/Telecommunications
Software/test.txt'
5. return FileResponse(open(file_path, 'rb'), content_type="text/plain")
6.
```

This method represents a response that returns the contents of a specified file. We display the contents of the test.txt file present on my computer. This gives :

