Riga Technical University

Faculty of Telecommunications and Electronics

ASSIGNMENT THREE

Python cloud full stack development

Student: Rofaida Nezzar.

Professor: Tianhua Chen.

Riga 2024

Example 1:

The **Django Hello World Project** is a basic web application developed to demonstrate the implementation of a simple "Hello, World!" page using Django, a Python-based web framework. The purpose of this project was to create an interactive webpage that combines Django's backend capabilities with frontend technologies like HTML, CSS, and JavaScript. This example showcases Django’s ability to manage templates and render web pages dynamically.

The project began with setting up the development environment. A virtual environment was created using the venv module to isolate dependencies, and Django was installed within this environment. After installing Django, a new project named my_project was initialized, and a Django app called hello was created to manage the logic for the "Hello, World!" feature. The app was then registered in the project's settings.py file, enabling Django to recognize it.

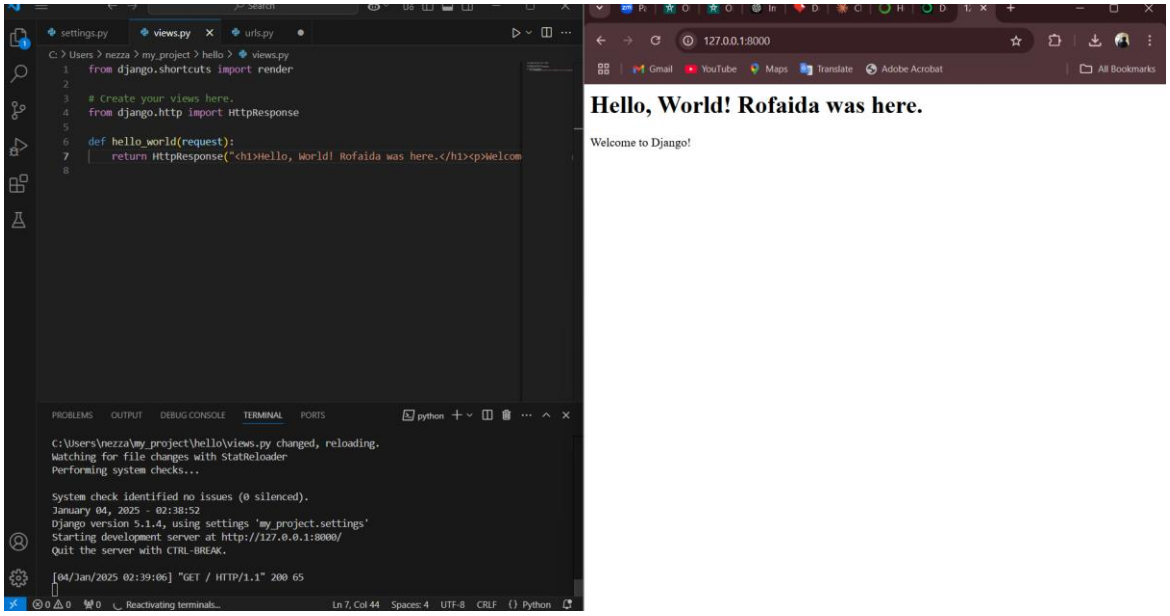
In the hello app, a view function named hello_world was defined in views.py. This function handles HTTP requests and renders the "Hello, World!" page using Django's template system. The URL routing was configured in the my_project/urls.py file, where a path was mapped to the hello_world view. This ensured that users visiting the /hello/ URL would see the desired webpage.

To design the frontend, a directory structure for templates was created within the hello app. The main HTML file, hello.html, was built with a simple yet elegant design. A pastel gradient background was added using CSS, and a clickable button was styled with hover effects and animations. JavaScript was used to add interactivity, displaying an alert message when the button was clicked. These frontend enhancements helped make the webpage visually appealing and functional.

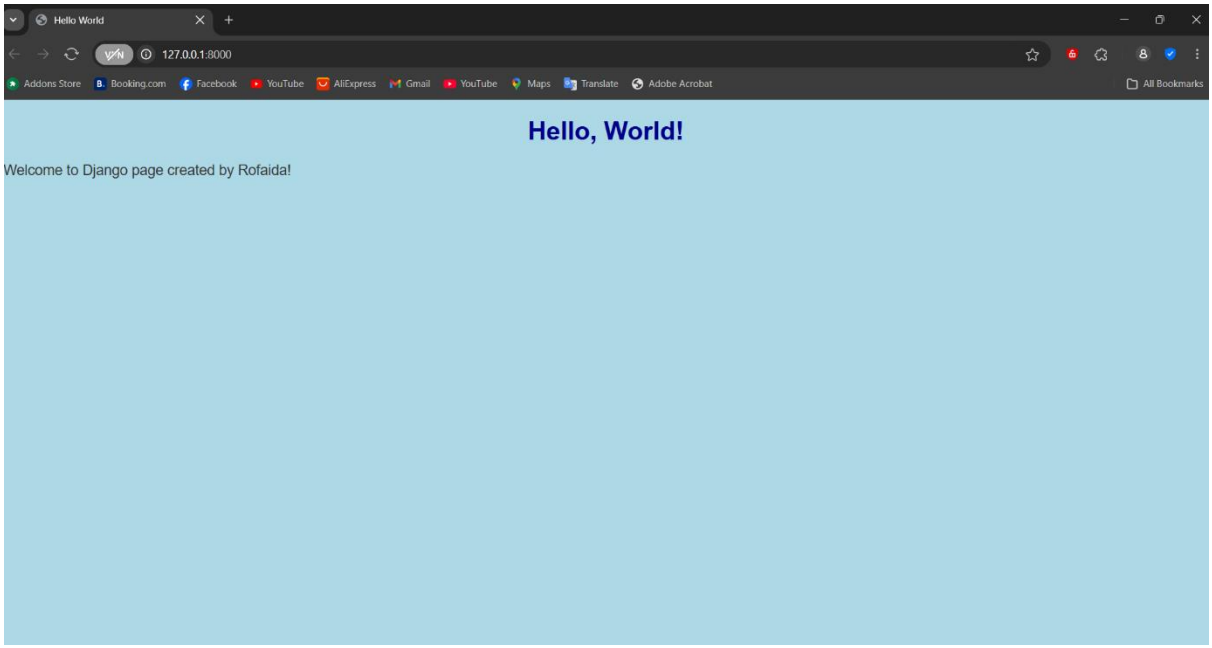
Once the development was complete, the Django development server was started using the command python manage.py runserver. The application was tested by visiting http://127.0.0.1:8000/hello/ in a web browser. The page successfully displayed the "Hello, World!" message, styled with CSS, along with the interactive button.

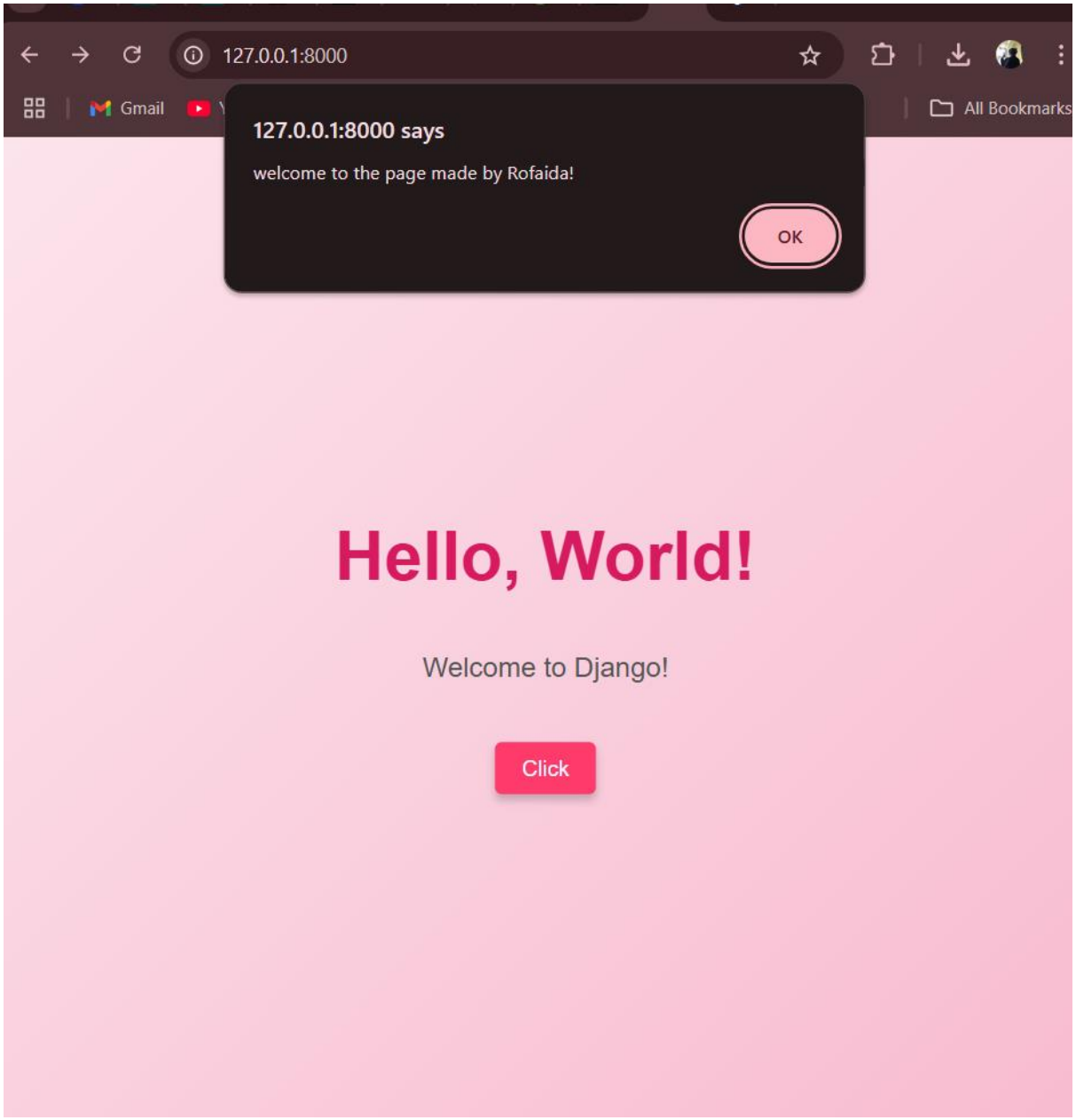
Note: all code was pushed to the github link.

The result of the web page:



At first the page was as simple as this, afterwards, I started playing with colours and design as I wanted it to be fun.





This is the final web page I made, pink and displays my name!

Example 2:

In this task we're asked to implement MTV pattern.

What is the MVT Pattern?

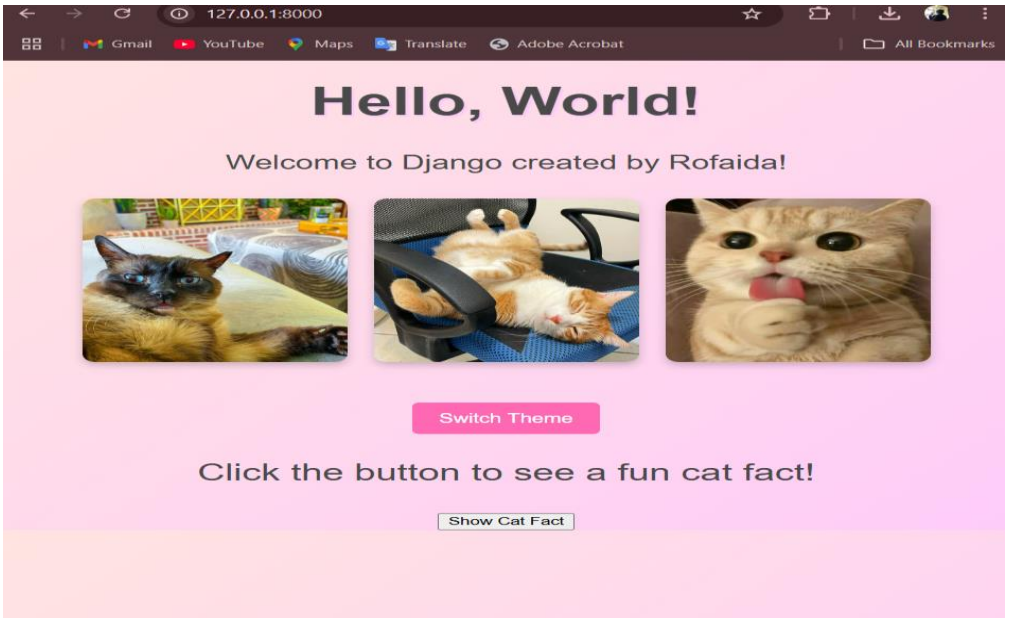
- **Model**: Handles data and database interactions (optional in this case since we aren't using a database yet).
- **View**: Processes user requests and passes data to the template.

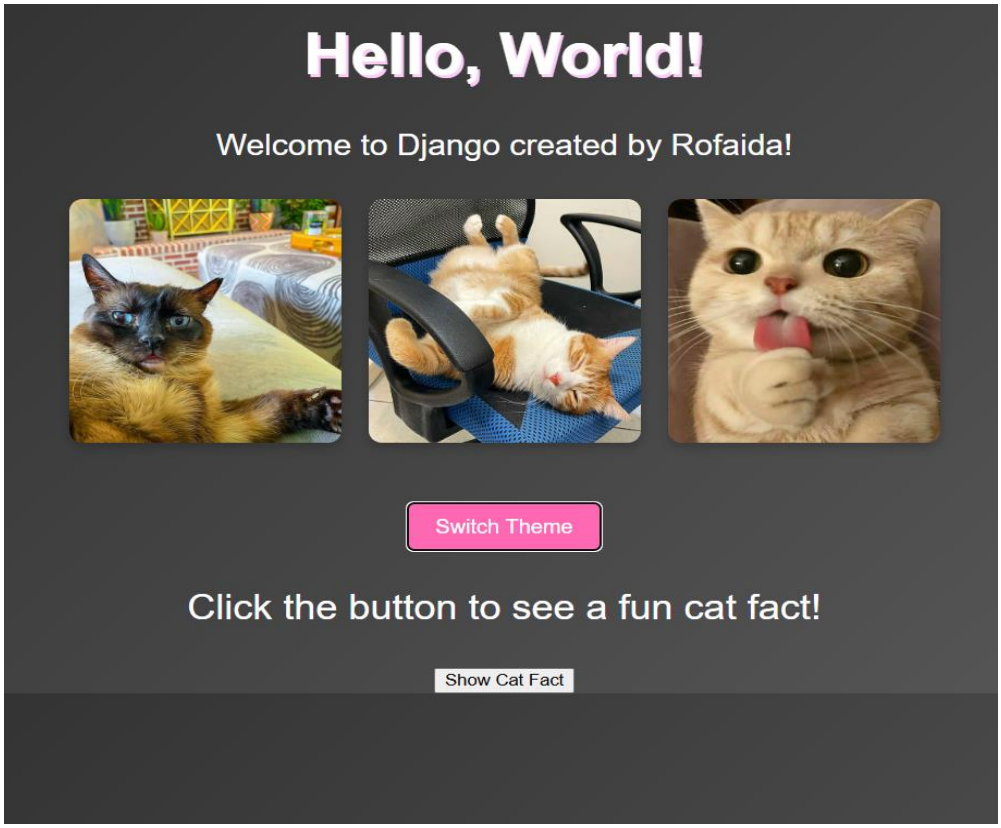
- **Template**: Renders the HTML page with data from the view.

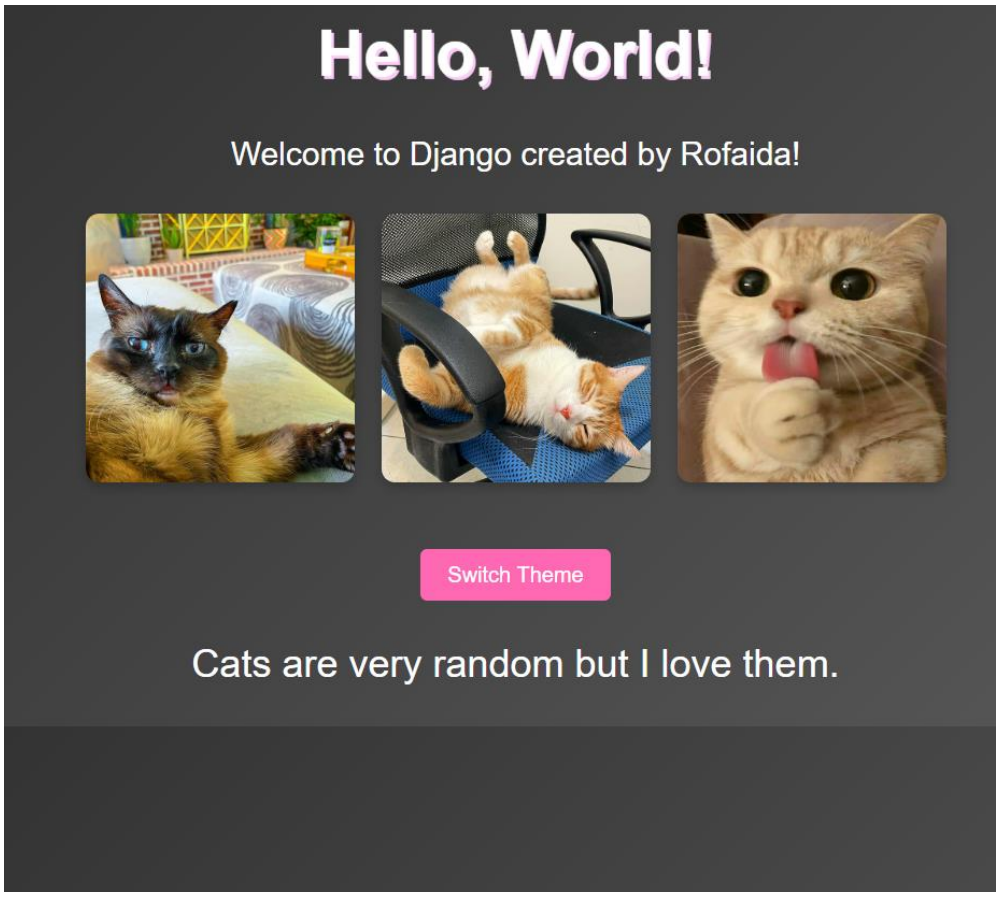
In this example, we built an improved version of a Django web application using the Model-View-Template (MVT) pattern. We enhanced the project by creating a dynamic and interactive user experience. Our page includes a gallery of adorable cat images, which we integrated into the app by creating a static folder and managing resources locally. To make the application more engaging, we added a theme-switching feature that allows users to toggle between a pastel light mode and a sleek dark mode. We also included a fun and educational element by dynamically displaying random fun facts about cats (because I love them), which change with a button click. For further creativity, we animated the page title and added fireworks animations to celebrate interactions, creating a lively and enjoyable atmosphere.

We added the static files to the project by first creating a static folder inside the hello app. This folder serves as the central location to store all static resources like images, CSS, and JavaScript files. After creating the folder, we placed the cat images directly into the hello/static/hello subdirectory. To ensure Django could locate and serve these files, we configured the STATICFILES_DIRS setting in the settings.py file. This allowed Django to recognize the static directory within the app. Finally, in the HTML template (hello.html), we used the {% load static %} template tag to reference and load the images dynamically. This setup enabled us to seamlessly include and display the static images in our gallery, adding charm and visual appeal to the page.

Images of the web:




Hello, World!
Welcome to Django created by Rofaida!
Switch Theme
Click the button to see a fun cat fact!
Show Cat Fact


Hello, World!
Welcome to Django created by Rofaida!
Switch Theme
Cats are very random but I love them.

Example 3 Cloud Message Board:

In this task, we developed a Cloud Message Board using Django to implement functionality for submitting and retrieving messages between users. This example showcased dynamic web application development with database integration and JSON response handling.

The goal of this example was to create a message board application with the following features:

1. **Submit Message Function:**
    - Users can set their name (A), specify a recipient (B), and leave a message for B.
    - The message is saved to the cloud (database).
2. **Get Messages Function:**
    - Users can retrieve the latest 20 messages they sent to specific recipients.
    - These messages are displayed dynamically.

We continued working within the same my project environment and created a new app called message board, we Implemented three main views in messageboard/views.py:

- **Index View:** Rendered the main HTML page.
- **Submit Message:** Accepted user input (sender, recipient, message) via POST request and saved it to the database.
- **Get Messages:** Retrieved the latest 20 messages for a specific sender and returned them as JSON.

URLs and Routing:

- Configured messageboard/urls.py to handle routes:
    - /messageboard/ for the main page.
    - /messageboard/submit/ for submitting messages.
    - /messageboard/get/<sender_name>/ for retrieving messages.
- All code is pushed in GitHub link.

Results:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ···    python

PS C:\Users\nezza\my_project> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
January 05, 2025 - 02:52:39
Django version 5.1.4, using settings 'my_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[05/Jan/2025 02:52:42] "GET /messageboard/ HTTP/1.1" 200 2356
[05/Jan/2025 02:52:52] "POST /messageboard/submit/ HTTP/1.1" 200 40
Bad Request: /messageboard/submit/
```

127.0.0.1:8000/messageboard/submit/

Gmail YouTube Maps Translate Adobe Acrobat All Bookmar

Pretty-print

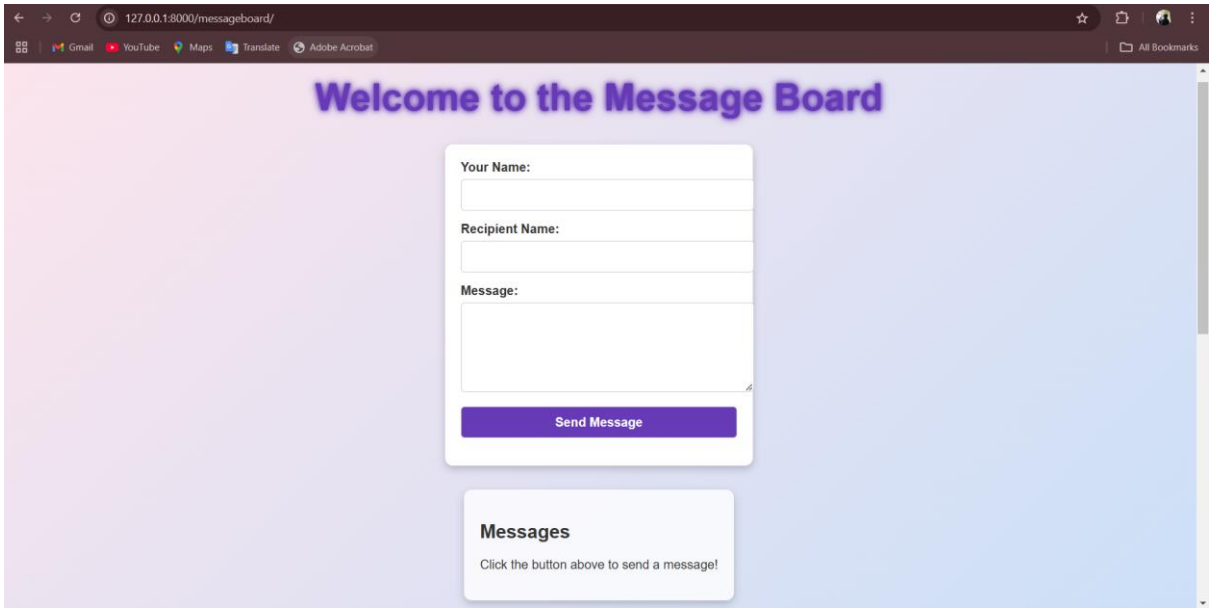

```
{"status": "Message sent successfully!"}
```


127.0.0.1:8000/messageboard/get/rofaida/

Gmail YouTube Maps Translate Adobe Acrobat All Bookmarks

Pretty-print


```
{"messages": [{"recipient": "nero", "message": "hola", "timestamp": "2025-01-
05T00:52:52.760Z"}]}
```

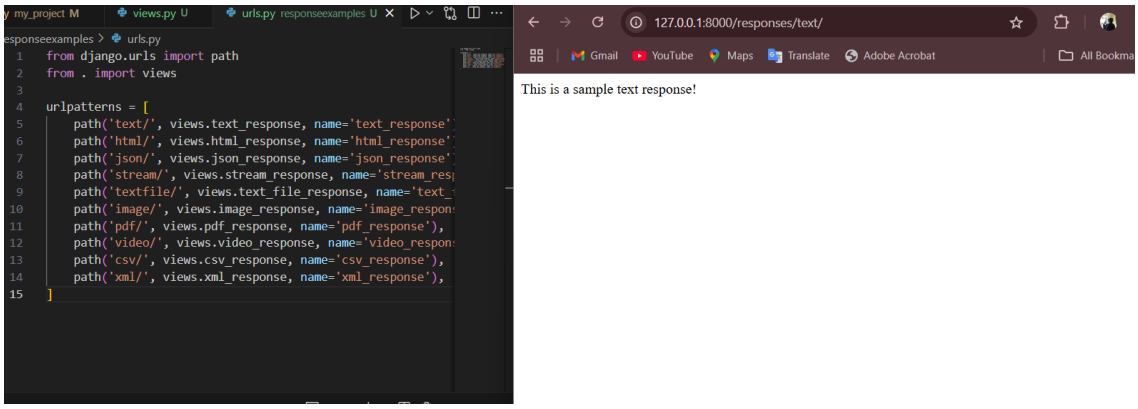


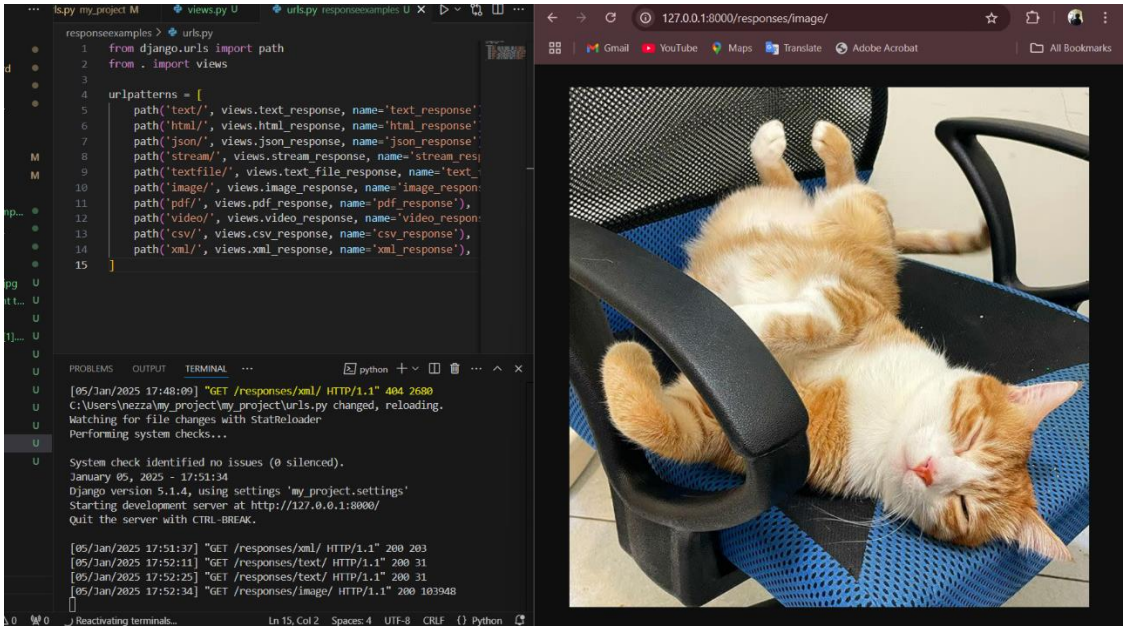
Example 4: Implementing Different Django Response Types

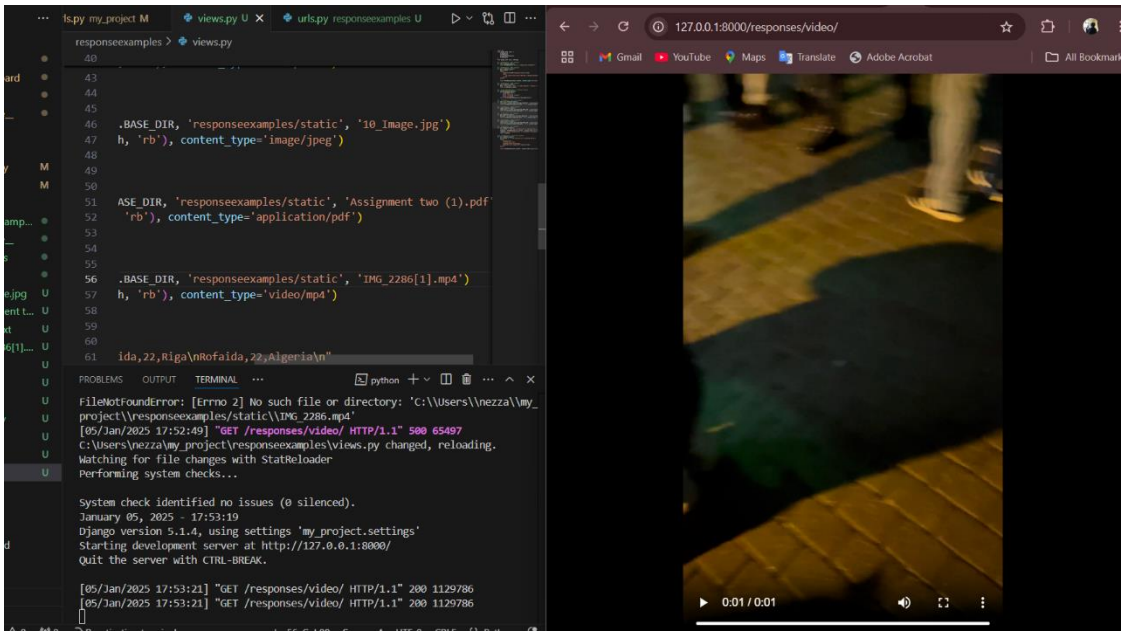
In this example, we explored various response types provided by Django to handle diverse content formats and demonstrate the framework's flexibility in delivering web content. This implementation involved defining and configuring the following response types:

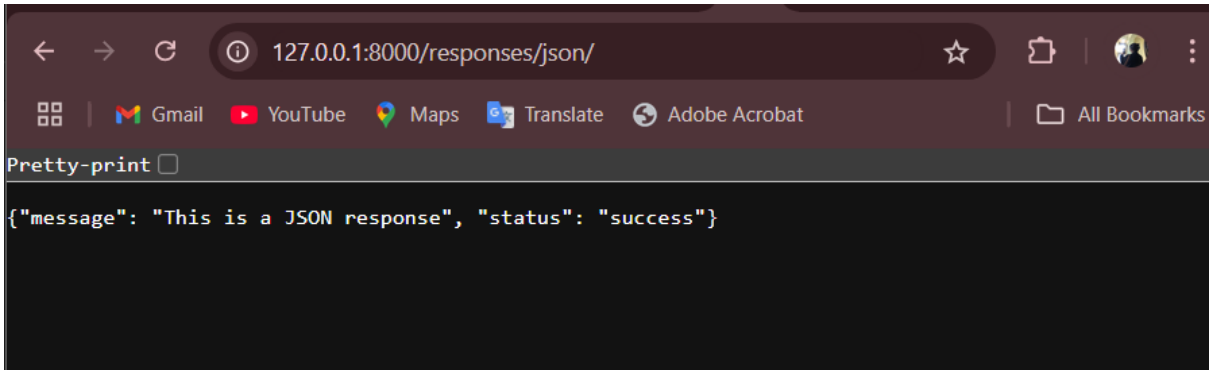
1. Text Response: http://127.0.0.1:8000/responses/text/
2. HTML Response: http://127.0.0.1:8000/responses/html/
3. JSON Response: http://127.0.0.1:8000/responses/json/
4. Streaming Response: http://127.0.0.1:8000/responses/stream/
5. Text File Response: http://127.0.0.1:8000/responses/textfile/
6. Image Response: http://127.0.0.1:8000/responses/image/
7. PDF Response: http://127.0.0.1:8000/responses/pdf/
8. Video Response: http://127.0.0.1:8000/responses/video/
9. CSV Response: http://127.0.0.1:8000/responses/csv/
10. XML Response: http://127.0.0.1:8000/responses/xml/

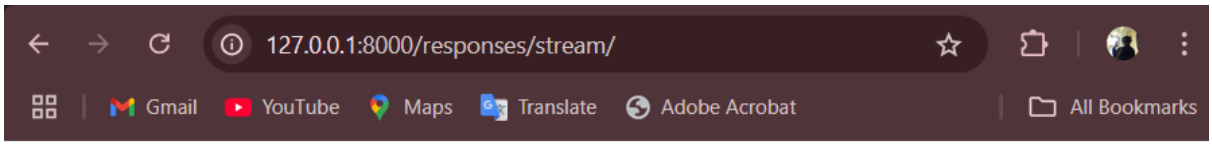
Screenshots:



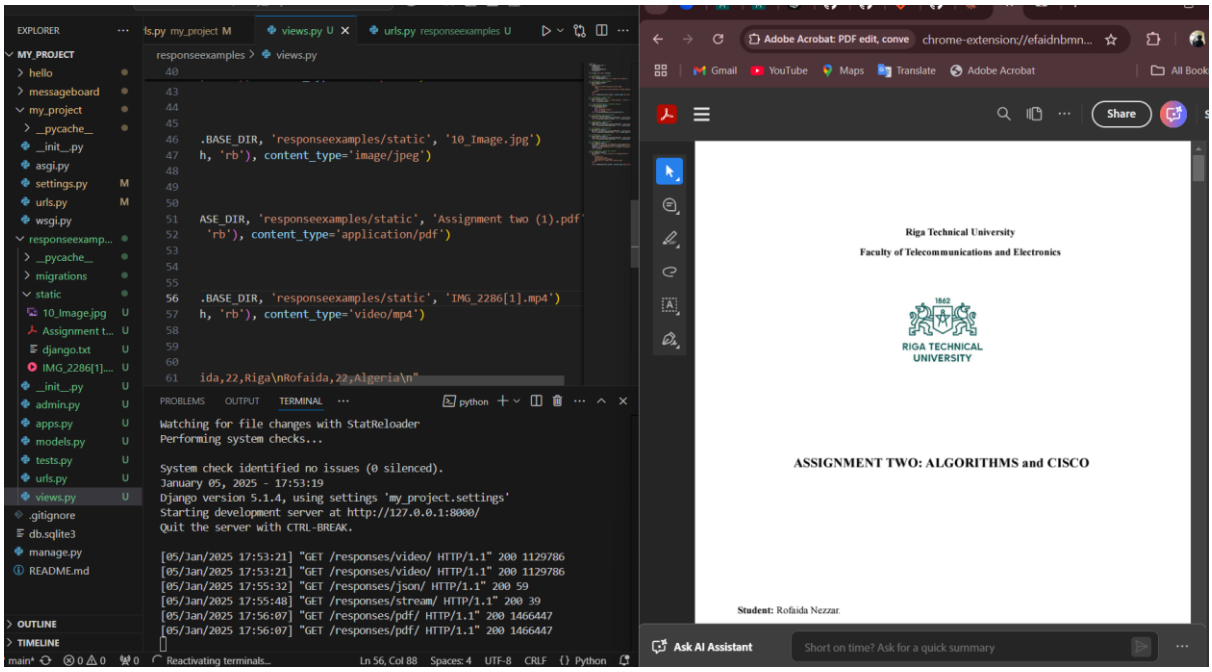


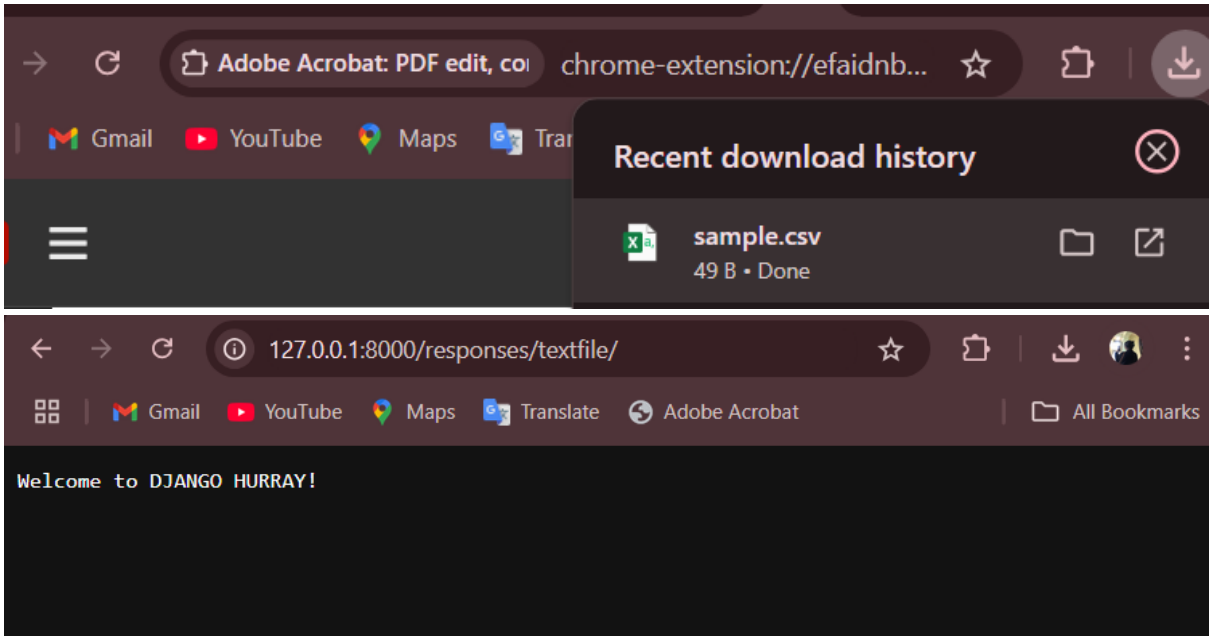






This is a streaming response in Django.



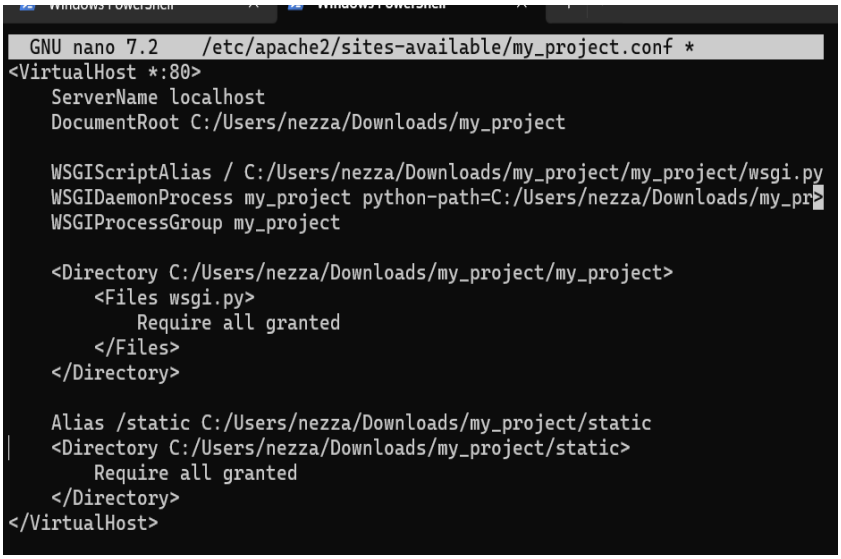


This example showcases Django's ability to handle a wide variety of response types, from simple text and JSON to streaming data and media files. By implementing these response types, we demonstrated how to provide diverse and dynamic content effectively. The knowledge gained through this example highlights the versatility of Django in building robust web applications capable of serving different types of data efficiently.

Task 2: Network Traffic Capture and Analysis (Above-mentioned examples)

First scenario HTTP + Apache: Configure Django to serve over HTTP with Apache:

Configuration:



```
  GNU nano 7.2    /etc/apache2/sites-available/my_project.conf *
<VirtualHost *:80>
    ServerName localhost
    DocumentRoot C:/Users/nezza/Downloads/my_project

    WSGIScriptAlias / C:/Users/nezza/Downloads/my_project/my_project/wsgi.py
    WSGIDaemonProcess my_project python-path=C:/Users/nezza/Downloads/my_pr
    WSGIProcessGroup my_project

    <Directory C:/Users/nezza/Downloads/my_project/my_project>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>

    Alias /static C:/Users/nezza/Downloads/my_project/static
    <Directory C:/Users/nezza/Downloads/my_project/static>
        Require all granted
    </Directory>
</VirtualHost>
```

tcp.port==80

No.	Time	Source	Destination	Protocol	Length
1064	48.932927	172.16.19.52	208.95.112.1	TCP	66
1067	48.991426	208.95.112.1	172.16.19.52	TCP	66
1068	48.991836	172.16.19.52	208.95.112.1	TCP	54
1070	48.994353	172.16.19.52	208.95.112.1	HTTP	232
1072	49.015040	208.95.112.1	172.16.19.52	TCP	56
1076	49.083299	208.95.112.1	172.16.19.52	HTTP/J…	539
1077	49.129022	172.16.19.52	208.95.112.1	TCP	54
1086	50.088570	172.16.19.52	208.95.112.1	TCP	55
1087	50.131961	208.95.112.1	172.16.19.52	TCP	56
1093	51.134821	172.16.19.52	208.95.112.1	TCP	55
1094	51.176085	208.95.112.1	172.16.19.52	TCP	56
1149	52.189896	172.16.19.52	208.95.112.1	TCP	55
1157	52.230191	208.95.112.1	172.16.19.52	TCP	56
1201	53.245017	172.16.19.52	208.95.112.1	TCP	55
1208	53.289722	208.95.112.1	172.16.19.52	TCP	56
1223	54.087077	172.16.19.52	208.95.112.1	TCP	54
1224	54.130832	208.95.112.1	172.16.19.52	TCP	56
1225	54.131174	172.16.19.52	208.95.112.1	TCP	54

```
Frame 5: 54 bytes on wire (432 bits), 54
Ethernet II, Src: LiteonTechno_c4:96:b9
Internet Protocol Version 4, Src: 172.16
Transmission Control Protocol, Src Port:

0000  00 00 5e 00 01 3d d0 39  57 c4 96 b9
0010  00 28 e6 be 40 00 80 06  b6 87 ac 16
0020  dd 5f d0 f4 00 50 60 b3  18 20 24 4c
0030  00 00 2d f5 00 00
```

Wireshark · HTTP / Requests · Wi-Fi

Request Type	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
HTTP Requests by HTTP Host	59				0.0011	100%	0.0300	7.579
ip-api.com	2				0.0000	3.39%	0.0100	7.479
/json	2				0.0000	100.00%	0.0100	7.479
239.255.255.250:1900	57				0.0011	96.61%	0.0300	7.579
*	57				0.0011	100.00%	0.0300	7.579

Display filter: Enter a display filter ... Apply

Copy Save as... Close

Wireshark Observations:

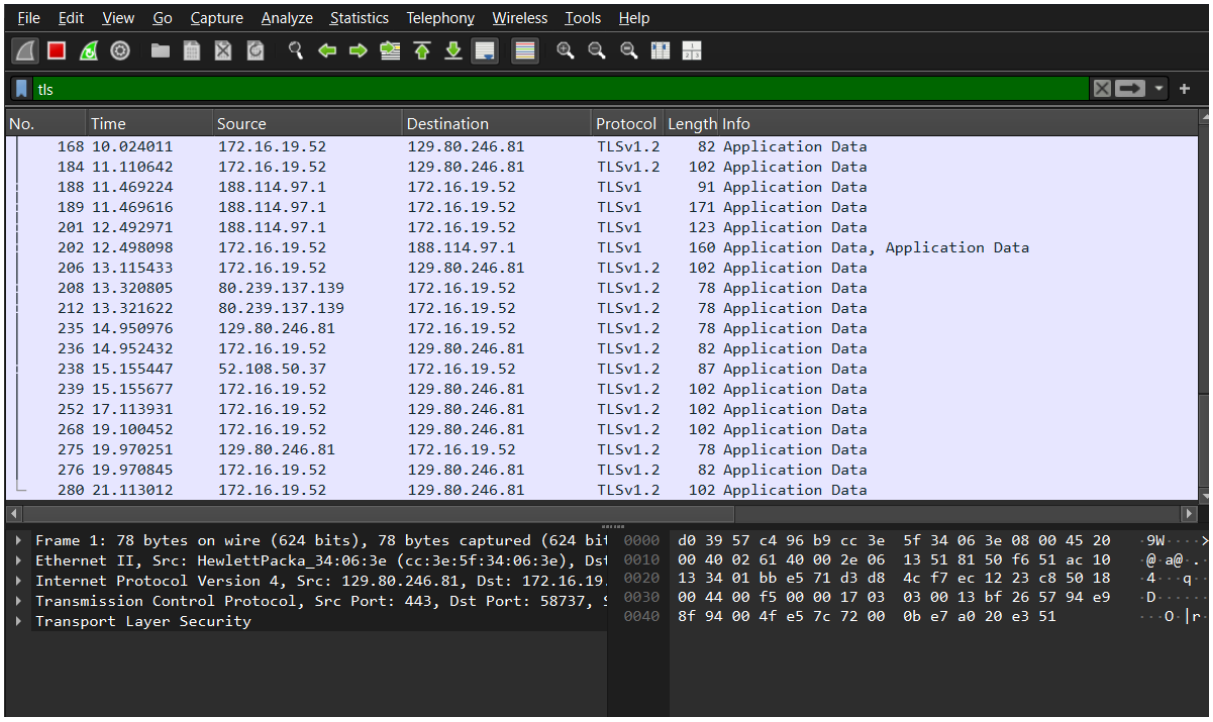
1. HTTP Requests and Responses:
    - The table shows HTTP requests made during your session.
    - The HTTP traffic indicates requests sent to specific destinations, such as ip-api.com and a multicast IP (239.255.255.250).
    - This reflects standard HTTP communication over port 80, which is unencrypted.
2. Packet Analysis:
    - Protocols: HTTP traffic is visible because the protocol is unencrypted.
    - TCP Packets: The HTTP communication relies on TCP, as shown in the packet details.

Scenario 2: HTTP + TLS + Apache

In this scenario we will configure Apache to serve Django over HTTPS using TLS:

Configuration:

```
</VirtualHost>

<VirtualHost *:443>
    ServerName localhost
    DocumentRoot C:/Users/nezza/Downloads/my_project

    WSGIScriptAlias / C:/Users/nezza/Downloads/my_project/my_project/wsgi.py
    WSGIDaemonProcess my_project python-path=C:/Users/nezza/Downloads/my_pr>
    WSGIProcessGroup my_project

    <Directory C:/Users/nezza/Downloads/my_project/my_project>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>

    Alias /static C:/Users/nezza/Downloads/my_project/static
    <Directory C:/Users/nezza/Downloads/my_project/static>
        Require all granted
    </Directory>

    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/localhost.crt
    SSLCertificateKeyFile /etc/ssl/private/localhost.key
</VirtualHost>
```



The captured traffic in Scenario 2 demonstrates encrypted communication using the **TLSv1.2 protocol**, which secures data transmission over HTTPS. Unlike HTTP in Scenario 1, where data was transmitted in plaintext, TLS encrypts the payload, ensuring confidentiality, integrity, and authentication. This is evident from the "Application Data" packets in Wireshark, which are unreadable without decryption keys. The traffic uses port 443, standard for HTTPS, and the TLS handshake establishes a secure connection through steps like **ClientHello** (offering cipher suites), **ServerHello** (selecting a cipher suite and sharing the server certificate), and key exchange to generate a session key for encryption. TLS advantages include preventing data tampering and protecting sensitive information like login credentials during transmission. Compared to HTTP, which exposes data in plaintext, TLS ensures robust security through symmetric and public-key encryption. This scenario highlights the critical role of TLS in securing modern web communication.