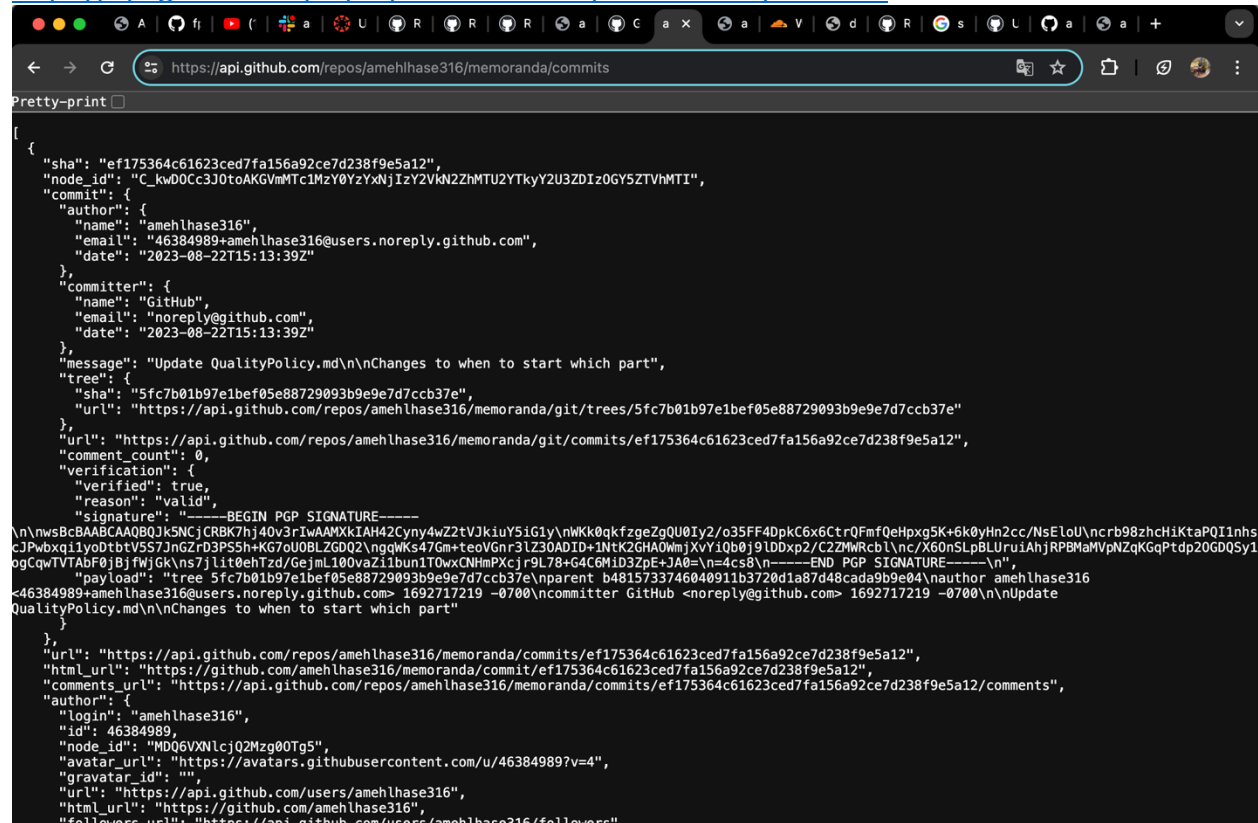


Assignment 2

Upper Layers of the OSI-Model

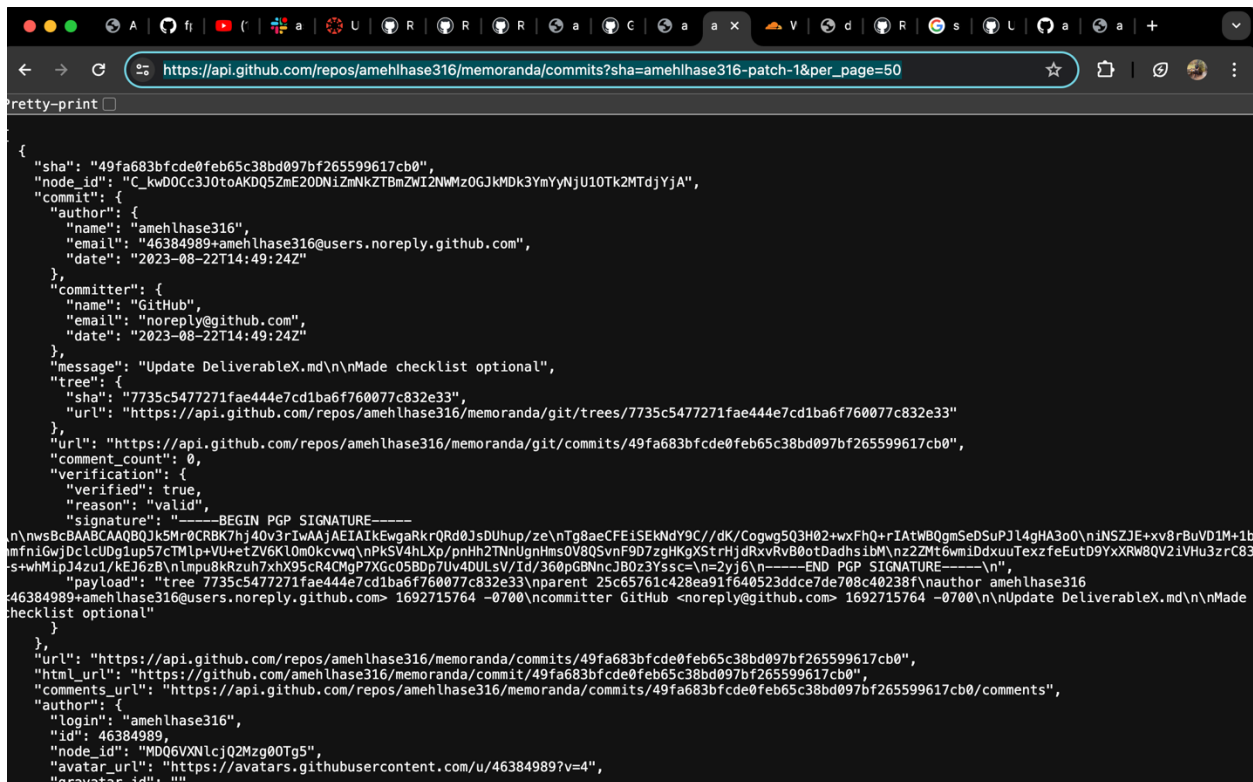
1 Understanding HTTP

<https://api.github.com/repos/amehlhase316/memoranda/commits>



```
[
  {
    "sha": "ef175364c61623ced7fa156a92ce7d238f9e5a12",
    "node_id": "C_kwD0Cc3J0toAKGVmMTc1MzY0YzYxNjIzY2VkN2ZhMTU2YTkyY2U3ZDIzOGY5ZTVhMTI",
    "commit": {
      "author": {
        "name": "amehlhase316",
        "email": "46384989+amehlhase316@users.noreply.github.com",
        "date": "2023-08-22T15:13:39Z"
      },
      "committer": {
        "name": "GitHub",
        "email": "noreply@github.com",
        "date": "2023-08-22T15:13:39Z"
      },
      "message": "Update QualityPolicy.md\n\nChanges to when to start which part",
      "tree": {
        "sha": "5fc7b01b97e1bef05e88729093b9e9e7d7ccb37e",
        "url": "https://api.github.com/repos/amehlhase316/memoranda/git/trees/5fc7b01b97e1bef05e88729093b9e9e7d7ccb37e"
      },
      "url": "https://api.github.com/repos/amehlhase316/memoranda/git/commits/ef175364c61623ced7fa156a92ce7d238f9e5a12",
      "comment_count": 0,
      "verification": {
        "verified": true,
        "reason": "valid",
        "signature": "-----BEGIN PGP SIGNATURE-----\n\nwsBcBAABCAAQ0JkSNCjCRBK7h40v3rIwAAMXkIAH42Cyny4wZ2tVJkiuY5iG1y\nwKk0qkfzgeZgQ0Iy2/o35FF4DpkC6x6CtrQFmf0eHpxg5K+6k0yHn2cc/NsEl\noU\\ncrb98zhcHiKtaPQI1nhs\n\nogCqWTVTAbF0jBjfwGK\\ns7jlit0ehTzd/Gejml100vaZilbun1T0wxCNHmPXcjr9L78+G4C6M1D3ZpE+JA0=\n\n-----END PGP SIGNATURE-----\n",
        "payload": "tree 5fc7b01b97e1bef05e88729093b9e9e7d7ccb37e\nparent b4815733746040911b3720d1a87d48cada9b9e04\nauthor amehlhase316\n46384989+amehlhase316@users.noreply.github.com 1692717219 -0700\ncommitter GitHub <noreply@github.com> 1692717219 -0700\n\nUpdate\nQualityPolicy.md\n\nChanges to when to start which part"
      },
      "url": "https://api.github.com/repos/amehlhase316/memoranda/commits/ef175364c61623ced7fa156a92ce7d238f9e5a12",
      "html_url": "https://github.com/amehlhase316/memoranda/commit/ef175364c61623ced7fa156a92ce7d238f9e5a12",
      "comments_url": "https://api.github.com/repos/amehlhase316/memoranda/commits/ef175364c61623ced7fa156a92ce7d238f9e5a12/comments",
      "author": {
        "login": "amehlhase316",
        "id": 46384989,
        "node_id": "MDQ6VXNlcjQ2Mzg0TG5",
        "avatar_url": "https://avatars.githubusercontent.com/u/46384989?v=4",
        "gravatar_id": "",
        "url": "https://api.github.com/users/amehlhase316",
        "html_url": "https://github.com/amehlhase316",
        "followers_url": "https://api.github.com/users/amehlhase316/followers"
      }
    }
  ]
}
```

https://api.github.com/repos/amehlhase316/memoranda/commits?sha=amehlhase316-patch-1&per_page=50



```
{
  "sha": "49fa683bfcde0feb65c38bd097bf265599617cb0",
  "node_id": "C_kwD0Cc3J0toAKDQ5ZmE2ODNiZmVkZTBmZWJlZWZlMzOGJkMDk3YmYyNjU1OTk2MTdjYjA",
  "commit": {
    "author": {
      "name": "amehlhase316",
      "email": "46384989+amehlhase316@users.noreply.github.com",
      "date": "2023-08-22T14:49:24Z"
    },
    "committer": {
      "name": "GitHub",
      "email": "noreply@github.com",
      "date": "2023-08-22T14:49:24Z"
    },
    "message": "Update DeliverableX.md\\n\\nMade checklist optional",
    "tree": {
      "sha": "7735c5477271fae444e7cd1ba6f760077c832e33",
      "url": "https://api.github.com/repos/amehlhase316/memoranda/git/trees/7735c5477271fae444e7cd1ba6f760077c832e33"
    },
    "url": "https://api.github.com/repos/amehlhase316/memoranda/git/commits/49fa683bfcde0feb65c38bd097bf265599617cb0",
    "comment_count": 0,
    "verification": {
      "verified": true,
      "reason": "valid",
      "signature": "-----BEGIN PGP SIGNATURE-----\n\nwBwBcBAABQAAQ0B0jK5Mw0CRBK7hj40v3rIwAAjAETATkEvgarKcrQrd0JsDUhup/ze\\nTgBaeCFEiSEkIdY9C//dK/Cogwp5Q3H02+wxFhQ+rIAtWB0gmSeDSuPJ14gHA3o0\\n\\nNSZJE+xxv8BuVD1M+1b\nmfniGwjdCldDgIup57CTMlp+VUuetZV6Kl0m0kcwq\\nPkSV4hLXp/oniH2TnUgnHmsOV80SvnF0D7zgHkgXStRhjdRkxvRvB8otbDadhsibM\\nz2ZMt6wmiDdxuutextz feEutD9YxXRW8QV2iVHu3zrC83\ns+whMip14zu1/KEJ6zB\\n\\nmpu8kRzuh7xhX95cR4QmP7XGc05B0p7Uv4DULsV/Id/360pG8NncJB0z3Yssc=\\n=2yj6\\n-----END PGP SIGNATURE-----\\n",
      "payload": "tree 7735c5477271fae444e7cd1ba6f760077c832e33\\nparent 25c65761c428ea91f640523ddce7de708c40238f\\nauthor amehlhase316\n46384989+amehlhase316@users.noreply.github.com> 1692715764 -0700\\ncommitter GitHub <noreply@github.com> 1692715764 -0700\\n\\nUpdate DeliverableX.md\\n\\nMade\nchecklist optional"
    }
  },
  "url": "https://api.github.com/repos/amehlhase316/memoranda/commits/49fa683bfcde0feb65c38bd097bf265599617cb0",
  "html_url": "https://github.com/amehlhase316/memoranda/commit/49fa683bfcde0feb65c38bd097bf265599617cb0",
  "comments_url": "https://api.github.com/repos/amehlhase316/memoranda/commits/49fa683bfcde0feb65c38bd097bf265599617cb0/comments",
  "author": {
    "login": "amehlhase316",
    "id": 46384989,
    "node_id": "MDQ6VXNlcjQ2Mzg0Tg5",
    "avatar_url": "https://avatars.githubusercontent.com/u/46384989?v=4",
    "gravatar_id": ""
  }
}
```

1. Explain the specific API calls you used, include the information you needed to provide, and include the link to the API documentation for that call.

URL: <https://api.github.com/repos/amehlhase316/memoranda/commits>

I used the following link to get all the commits on the default branch for a repository name memoranda. **Information Provided:** Username (**amehlhase316**) and Repository name (**memoranda**). This API call retrieves a list of commits on the default branch of the specified repository.
<https://docs.github.com/en/rest/commits/commits?apiVersion=2022-11-28#list-commits>

URL: https://api.github.com/repos/amehlhase316/memoranda/commits?sha=amehlhase316-patch-1&per_page=50

I used the following link to create the call that specifies a specific branch (not the default), and sets the per-page limit to 50. This API call retrieves a list of commits on the specified branch amehlhase316-patch-1 and sets the page limit to 50 commits per page. **Information Provided:** Username (**amehlhase316**), Repository name (**memoranda**), Branch (**amehlhase316-patch-1**), and Page limit (**50**)
<https://docs.github.com/en/rest/using-the-rest-api/using-pagination-in-the-rest-api?apiVersion=2022-11-28>

2. Explain the difference between stateless and stateful communication.

Stateless Communication:

- Each request from a client to a server must contain all the information the server needs to fulfill that request.
- No client context is stored on the server between requests.
- Example: HTTP is a stateless protocol. Each HTTP request is independent and contains all the information the server needs to process.

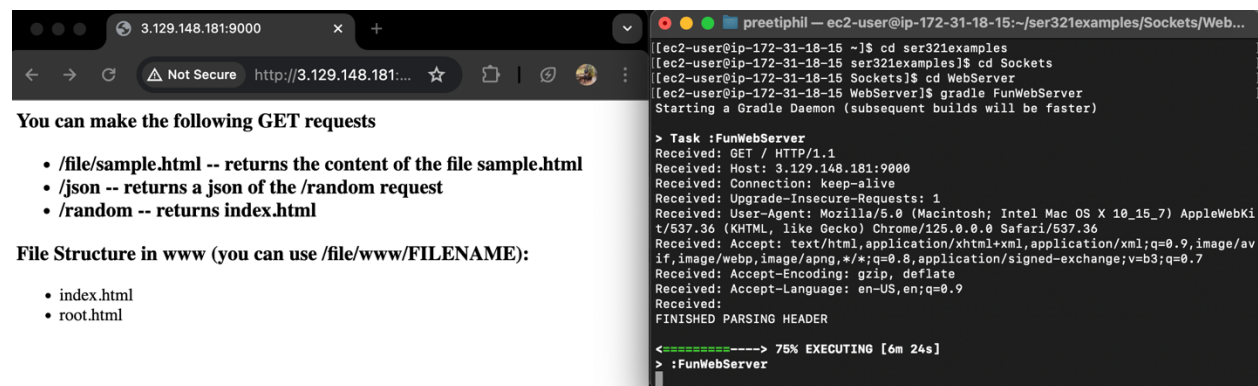
Stateful Communication:

- The server retains the client's state across multiple requests.
- Each request is dependent on the state stored on the server from previous interactions.
- Example: FTP (File Transfer Protocol) is stateful, where the server maintains the connection and session state throughout the communication process.

2 Set up your second system and run servers on it

2.1 Getting sample code onto your systems

2.2 Running a Simple Java WebServer

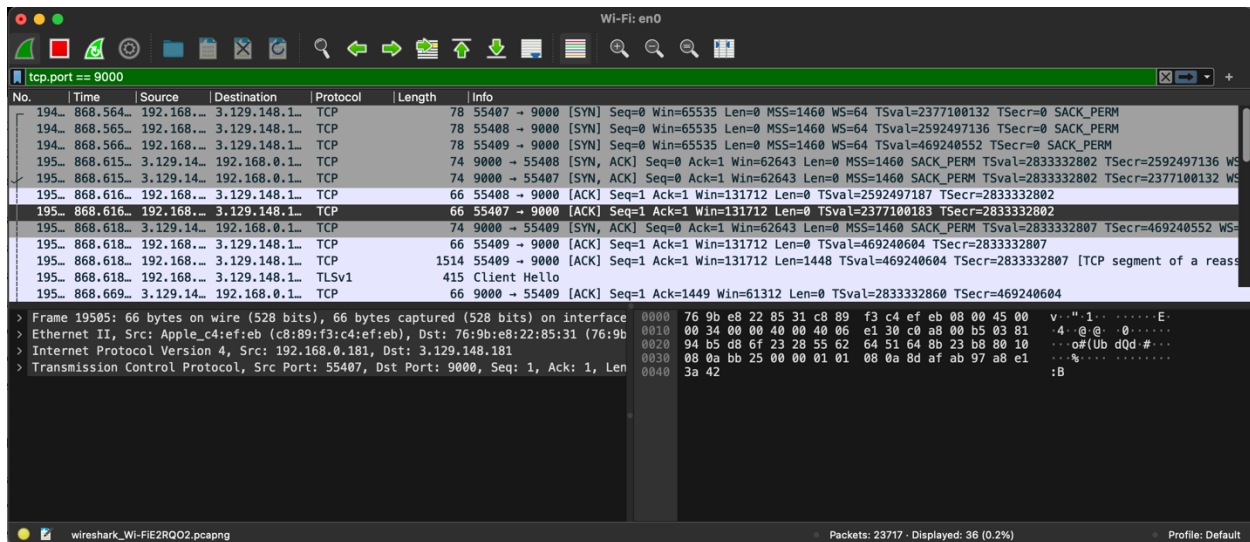


The screenshot displays a web browser window on the left and a terminal window on the right. The browser window shows a 'Not Secure' warning and a list of GET requests: `/file/sample.html`, `/json`, and `/random`. The terminal window shows the execution of a Java web server, with the following output:

```
[ec2-user@ip-172-31-18-15 ~]$ cd ser321examples
[ec2-user@ip-172-31-18-15 ser321examples]$ cd Sockets
[ec2-user@ip-172-31-18-15 Sockets]$ cd WebServer
[ec2-user@ip-172-31-18-15 WebServer]$ gradle FunWebServer
Starting a Gradle Daemon (subsequent builds will be faster)

> Task :FunWebServer
Received: GET / HTTP/1.1
Received: Host: 3.129.148.181:9000
Received: Connection: keep-alive
Received: Upgrade-Insecure-Requests: 1
Received: User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36
Received: Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Received: Accept-Encoding: gzip, deflate
Received: Accept-Language: en-US,en;q=0.9
Received: FINISHED PARSING HEADER
<=====75% EXECUTING [6m 24s]
> :FunWebServer
```

2.3 Analyze what happens



Wireshark should still be running in the background. Go to Wireshark, and create a filter that shows network traffic to and from your WebServer. Take a screenshot of your Wireshark capture and add it to your document.

Deliverable: In your document, answer the following (1-2 points each):

1. What filter did you use? Explain why you chose that filter.
I chose this filter because it isolates traffic to and from port 9000, which is where our FunWebServer is running. This filter allows us to focus specifically on the network packets related to our web server, eliminating unrelated traffic and making it easier to analyze interactions with the server.
2. What happens when you are on the /random page and click the "Random" button? Compare this to refreshing your browser. (You can also use the command line output that the WebServer generates to answer this.)
Clicking the "Random" Button: When you click the "Random" button on the /random page, a new request is sent to the server to generate and retrieve a new photo. This typically results in a new response from the server, which includes the random photo in the page content.

Refreshing the Browser: Refreshing the browser on the /random page causes the entire page to reload, sending a new HTTP GET request to the server. This action retrieves the latest version of the page, including a new random photo if it is generated server-side each time the page is loaded.

3. What types of response codes are you able to receive through different requests to your server?
200 OK: Indicates that the request was successful and the server has sent back the requested resource

404 Not Found: Indicates that the server cannot find the requested resource. This can occur if an invalid URL path is accessed.

4. Explain the response codes you receive and why you get them.

200 OK:

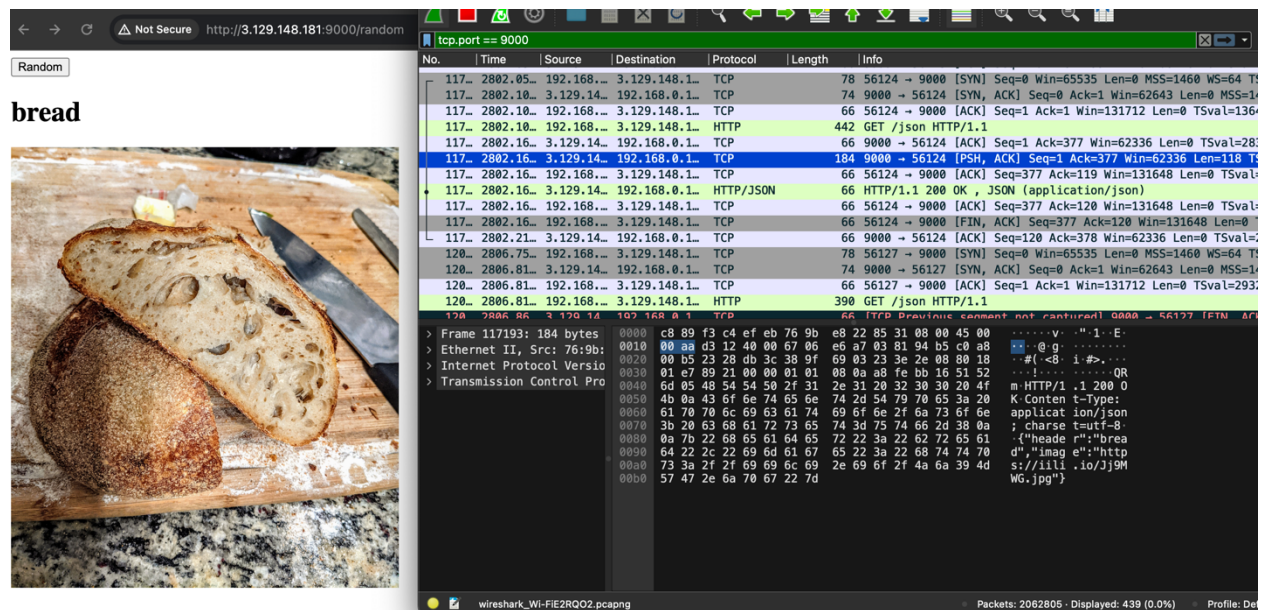
This code is received when the server successfully processes the request and returns the requested resource. For example, accessing the **/random** page or the homepage typically results in a 200 OK response.

404 Not Found:

This code occurs when a client requests a resource that does not exist on the server. For instance, accessing a non-existent URL like **/nonexistent** would result in a 404 Not Found response.

5. When you do a `<publicIPOfYourSecondMachine>:9000`, take a look at what Wire-shark generates as a server response. Are you able to find the data that the server sends back to you? (This should be the "Data" section of your response.)

Yes, as shown in the screenshot I can see that a bread image is viewed.



The screenshot displays a web browser window on the left and a Wireshark packet capture on the right. The browser shows a page titled "bread" with a large image of a loaf of bread. The address bar indicates the URL is `http://3.129.148.181:9000/random`. The Wireshark interface on the right shows a list of network packets. The selected packet is an HTTP 200 OK response from the server to the client. The packet details pane shows the response structure, including the status line `HTTP/1.1 200 OK` and the content type `application/json`. The packet bytes pane shows the raw data of the response, which includes a JSON object containing the image data.

6. Based on the previous question, explain why HTTPS is now more common than HTTP. HTTPS (HyperText Transfer Protocol Secure) is more common than HTTP because it provides encrypted communication between the client and the server. This encryption ensures data integrity, confidentiality, and authenticity, protecting against eavesdropping, man-in-the-middle attacks, and other security threats. HTTPS is essential for securing sensitive data, such as login credentials, personal information, and financial transactions.
7. In our case - what port does the server listen to for HTTP requests, and is that the most common port for HTTP?

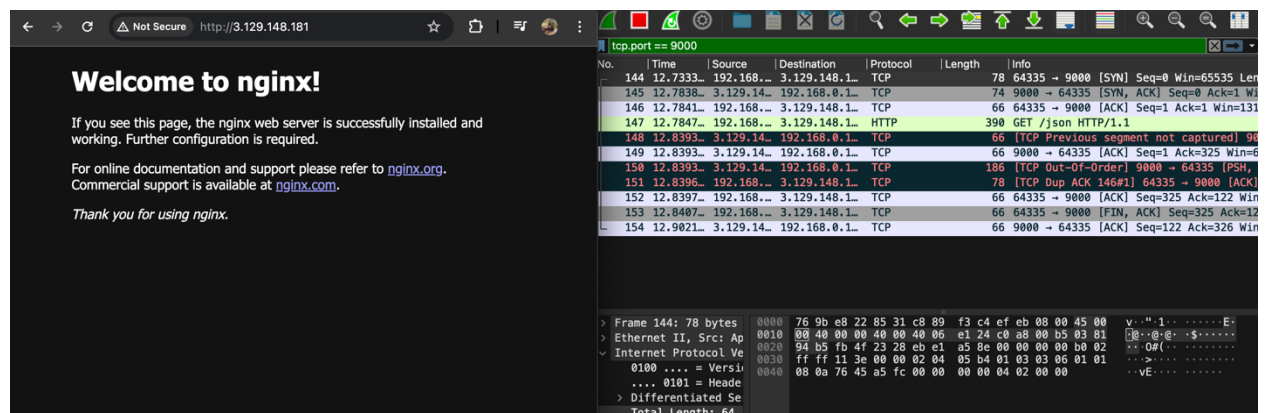
The server listens on port 9000 for HTTP requests in this setup. The most common port for HTTP is port 80. However, for this specific project, we configured the server to listen on port 9000, which is why our traffic is directed to that port.

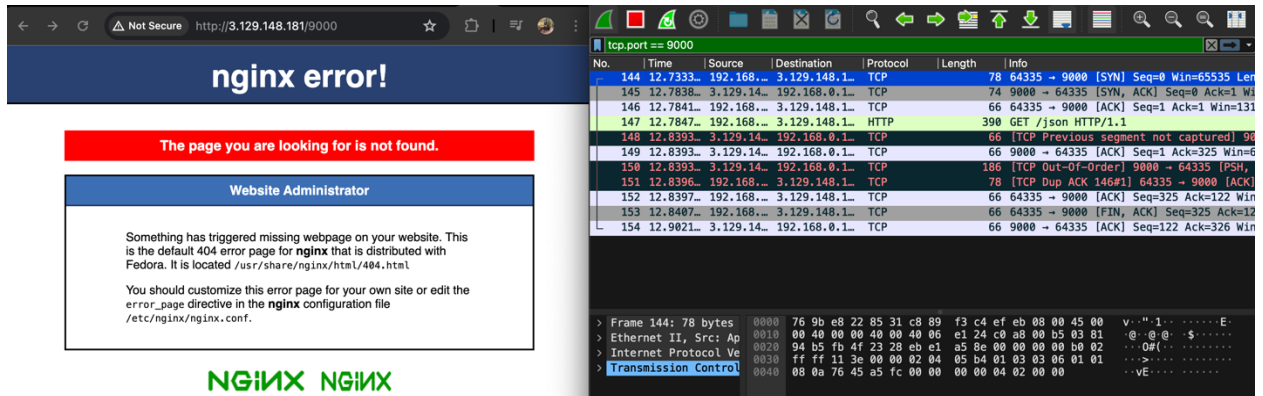
8. Which local port is used when sending different requests to the WebServer?

When sending different requests to the WebServer, the local machine (client) typically uses an ephemeral port. These are temporary ports allocated for the duration of the connection. The specific port number can vary with each request, and Wireshark can show these in the captured packets under the source port field as 55407, 55408, 55409.

2.4 Setting up a "real" WebServer

1. What is the URL that you can now use to reach the main page?
`http://3.129.148.181/`
2. Check the traffic to your WebServer. What port is the traffic going to now? Is it the same as before, or is it (and should it) be different?
The traffic is now going to port 80, which is the default HTTP port. Previously, the traffic was directed to port 9000. The traffic should be different because Nginx is now handling incoming requests on port 80 and proxying them to port 9000.
3. Is it still using HTTP, or is it now using HTTPS? Why?
It is still using HTTP. This is because we have not configured Nginx to use HTTPS, which requires additional steps such as obtaining an SSL certificate and updating the Nginx configuration to handle HTTPS traffic
4. Could you change your security settings on AWS now?
Yes, this involves modifying the inbound rules of the security group associated with your EC2 instance to allow HTTP traffic.
5. Take a screenshot of your web browser, your second machine, and the port number on Wireshark. This should be similar to the screenshot you took before (but also with Wireshark), and add it to your document for this task. Note: If we are unable to see that you reached the WebServer with the "different URL", we will not know if you actually set up the server correctly. Therefore, please make sure that it shows up if you want points for this task.





2.5 Setting up HTTPS

1. What port is your traffic going through now?

After setting up HTTPS, the traffic should be going through port **443**. This is the standard port for HTTPS traffic.

2. Can you still find the plain text responses that were found with HTTP?

With HTTPS enabled, the responses are no longer in plain text. HTTPS encrypts the data between the client and the server, so any HTTP traffic that was previously in plain text will now be encrypted. This includes headers, response bodies, and any other data transmitted over the connection

2.6 Some programming on your WebServer

2.6.1 Multiply

In the **multiply?** endpoint of the **WebServer** class, I decided to handle two scenarios: one where the required parameters **num1** and **num2** are missing, and another where these parameters are present but not valid integers.

For the case where parameters are missing, I set the HTTP response status code to **400 Bad Request** along with an appropriate error message. This is because the client's request is syntactically incorrect, and the server cannot process it due to missing parameters. The **400 Bad Request** status code indicates to the client that the request cannot be fulfilled due to bad syntax.

For the case where the parameters are present but not valid integers, I again set the HTTP response status code to **400 Bad Request** with an error message explaining that the provided parameters must be integers. This is because the server cannot perform the multiplication operation if the parameters are not valid integers

2.6.3 Make your own requests

1. `/wordcount?text=Hello%20world`

This request counts the number of words in the provided text "Hello world".

2. `/reverse?text=Hello%20world`

This request reverses the provided text "Hello world".