# **ICT171 Assignment 2: Cloud Server Project & Video Explainer**

**Name:** Anuj Mahato  
 **Student ID:** 35450857  
 **Domain Name:** [https://anuj.fun](https://anuj.fun/) **Server IP Address:** 170.64.253.234  
 **License Chosen:** MIT License  
 [**Github Link**](https://github.com/AJ-1220/ANUJ-ICT171)   
 [**Video Link**](https://youtu.be/HqxH3ka98Qo)

## **Table of Contents**

1. Project Scenario
2. Server Setup
3. SSH Connection
4. System Update & Upgrade
5. Install Prerequisites
6. Add Universe Repository
7. Add Certbot PPA
8. Install Certbot’s Nginx Package
9. Setup SSL/TLS with Certbot for Nginx
10. DNS Functionality and Accessibility
11. Web Server Configuration (Nginx)
12. Editing HTML Page
13. Final Website and Functionality
14. GitHub Repository Submission & Commits
15. Bash Scripting (server.sh)
16. Windows Scripting (File Transfer & Server Access)
17. Script Usefulness and Output
18. License Explanation
19. Video Demonstration & Script
20. References

## **1. Project Scenario**

This project involves deploying a cloud-based website specifically designed to provide free stock images for users to download and use. The website is aimed at students, bloggers, and anyone who needs high-quality images without worrying about copyright or licensing fees.. At first, I honestly had no idea how to even put a site on the cloud, but I learned by doing. The main thing was to show my ICT171 skills, so I set up a Linux server on DigitalOcean (never used it before), did all the DNS stuff to point my own domain there, and managed to get HTTPS running using Certbot. I built the whole website from scratch so it actually works for anyone on the internet.

While doing all this, I learned a lot about server admin, using Linux commands (sometimes messing it up), how DNS and SSL/TLS works, and how to write everything up proper for uni. Also figured out how to keep all my files and changes on GitHub which is good for backup. Now the website isn’t just for this unit, it can actually help someone and it’s there for other ICT171 students who wanna do something like this in future.

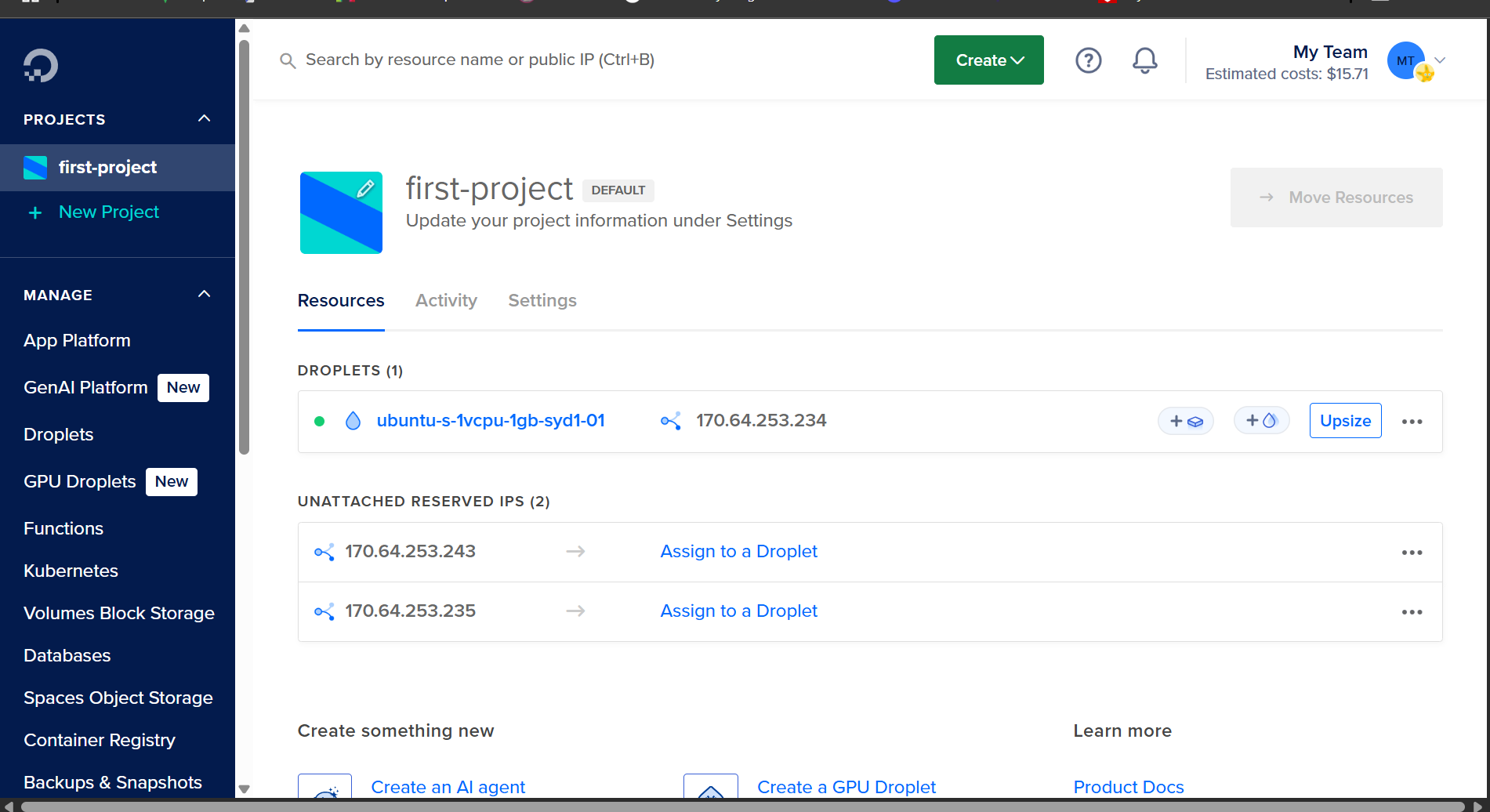
## **2. Server Setup**

**Provider:** DigitalOcean (IaaS)  
 **Operating System:** Ubuntu 22.04 LTS

**Steps Taken:**

* First, I signed up on DigitalOcean and created what they call a “droplet”, which is just their name for a mini server.
* I set it up so I had root access, so I could basically change whatever I want on it.
* DigitalOcean gave me a public IP (mine’s 170.64.253.234), so my server’s out there on the internet for anyone to find.
* After that, I registered my own domain – anuj.fun – and had to play around with the DNS settings until it pointed to my server’s IP. Took a bit of trial and error honestly.
* Once all that was sorted, it meant I was ready to start installing stuff and make the site live for anyone to check out.

**Screenshot:** DigitalOcean droplet dashboard and DNS management page, confirming the correct server setup and domain linkage.

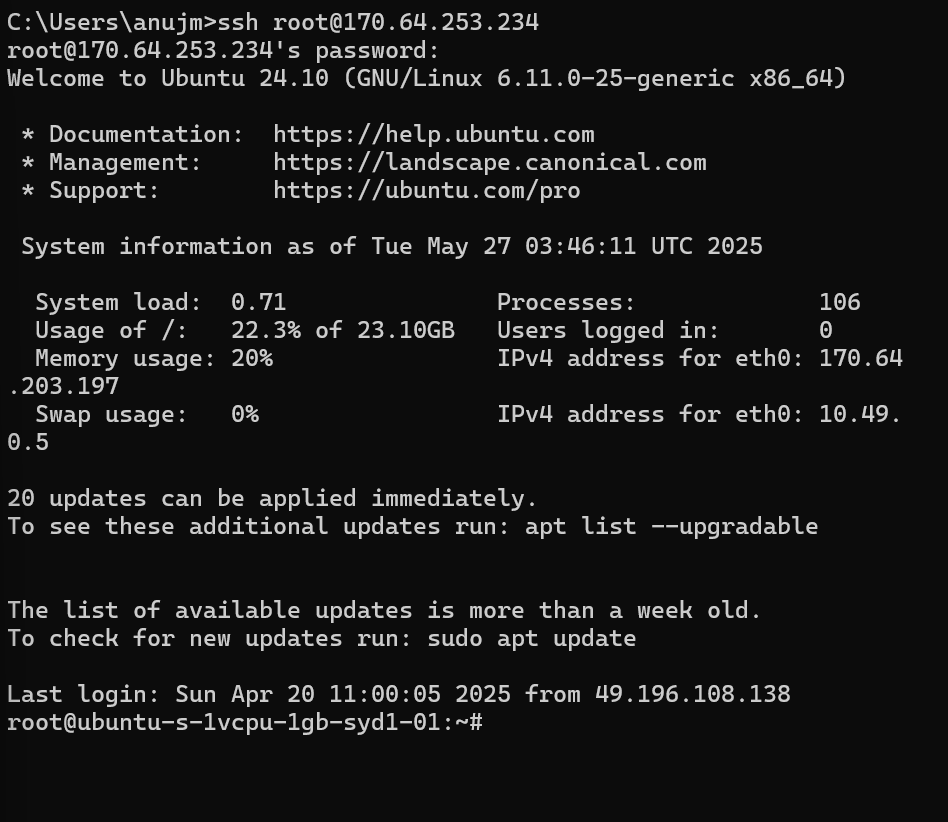


## **3. SSH Connection**

Secure Shell (SSH) is how I connected to my server from my own computer, and it's pretty much what everyone uses because it’s safe. I used SSH to log into my DigitalOcean server straight from my laptop’s terminal. Almost everything I did—like setting up, installing software, and making changes—was done through SSH. Without it, I really couldn’t have finished the project.

* The main command I used for connecting was: ssh root@170.64.253.234
* SSH was used for all the setup, updates, and fixing stuff whenever I messed up.
* All my server management happened in the terminal using SSH.

**Screenshot:** Terminal window showing a successful SSH connection to the cloud server as the root user.

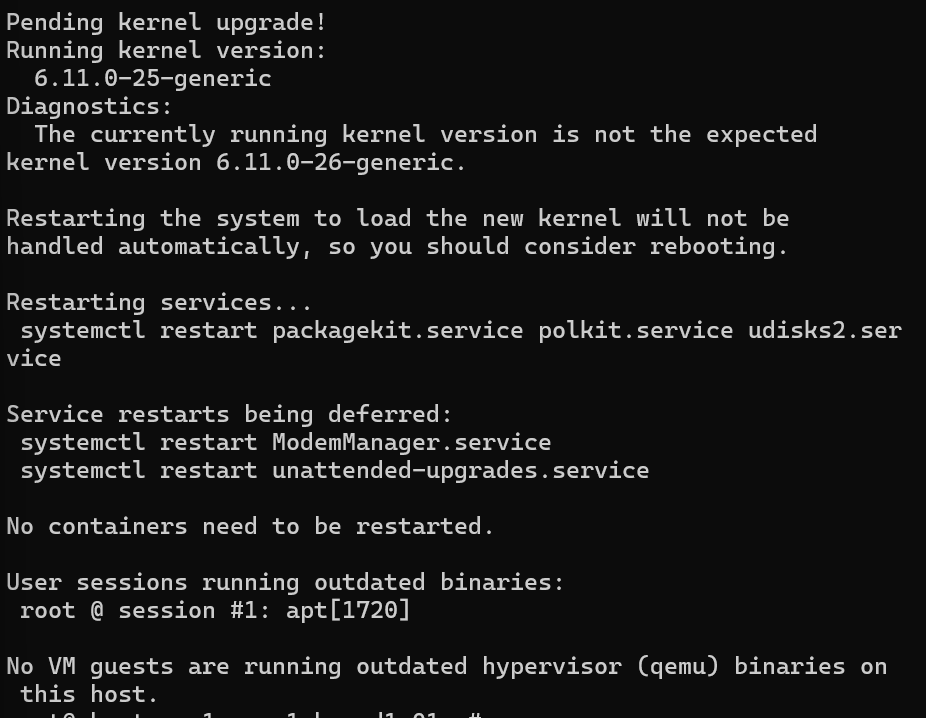
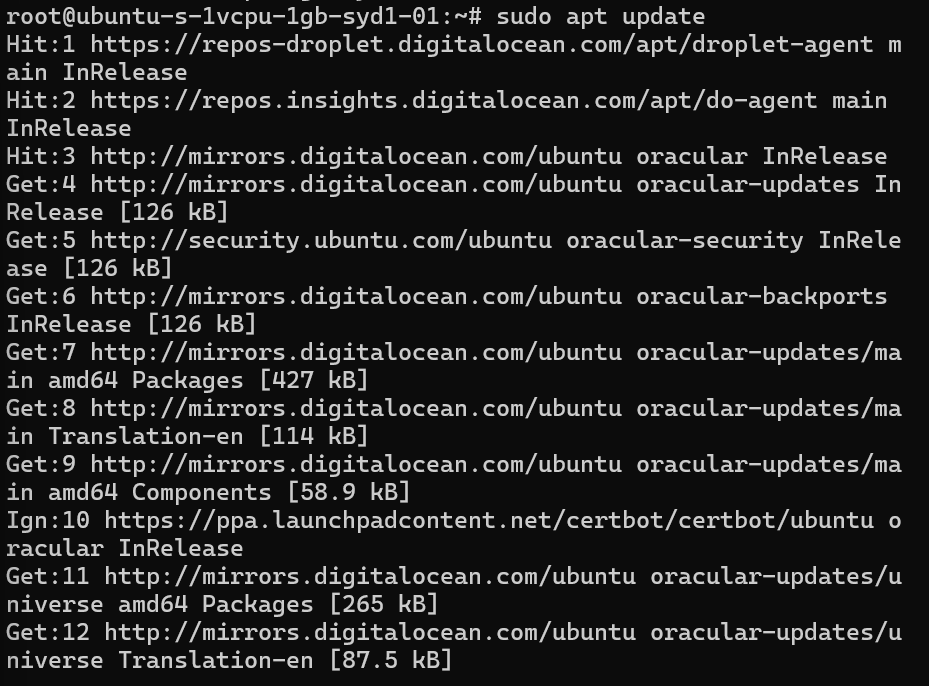


## **4. System Update & Upgrade**

As soon as I managed to get into the server with SSH, the first thing I did was make sure everything was updated and safe. This just means you gotta run some commands to update the package list and then upgrade all the stuff that’s installed, so you don’t have any old software that could mess things up or be insecure.

* I used: sudo apt update
* Then: sudo apt upgrade -y
* Doing this right away stops random problems later and makes sure the server is running the newest versions of everything.

**Screenshot:** Terminal output confirming updates and upgrades were completed successfully.

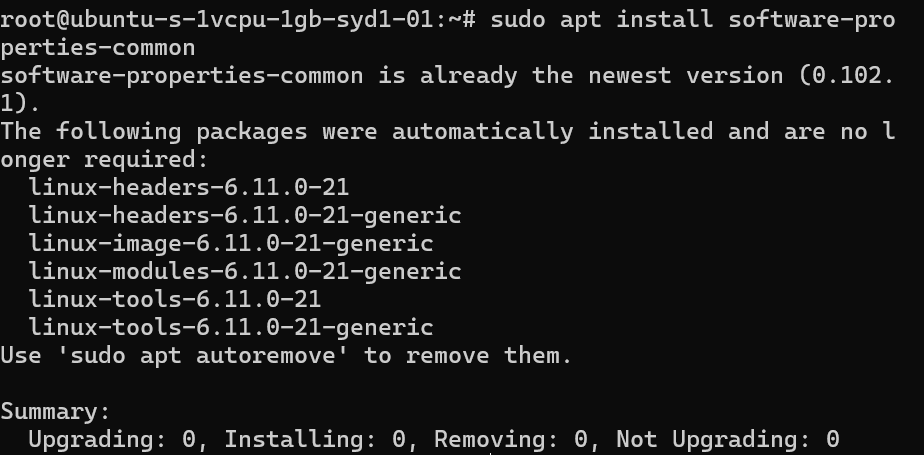


## **5. Install Prerequisites**

Before installing advanced packages, I installed software-properties-common which is required for managing additional repositories and installing third-party software on Ubuntu:

* sudo apt install software-properties-common

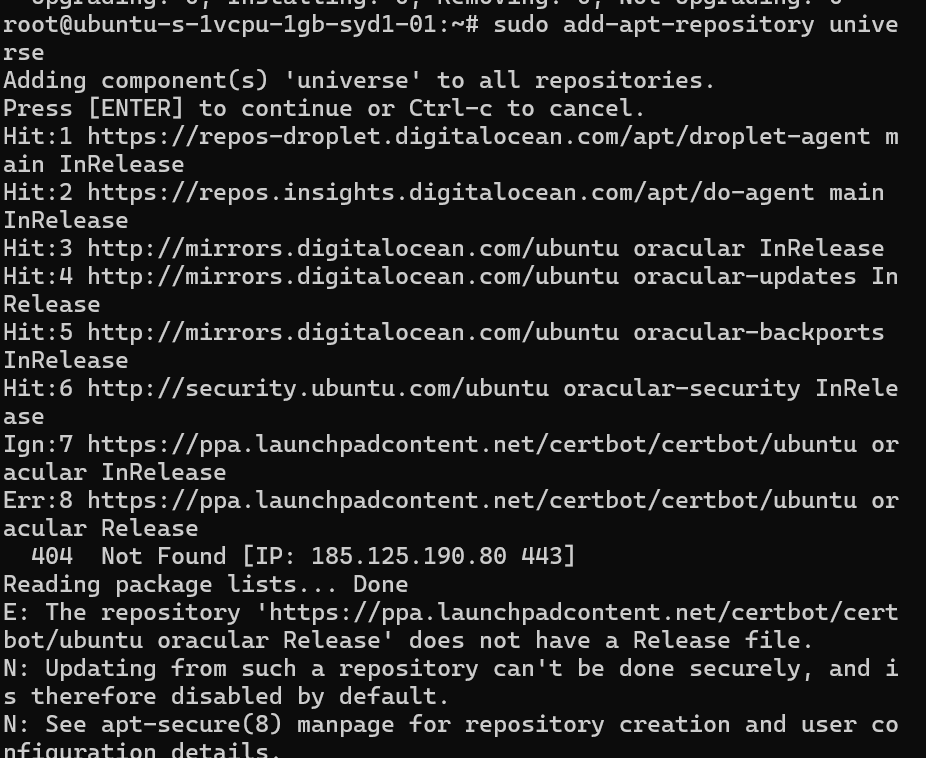
This package allows for the easy addition of repositories necessary for Certbot and other services later in the project.



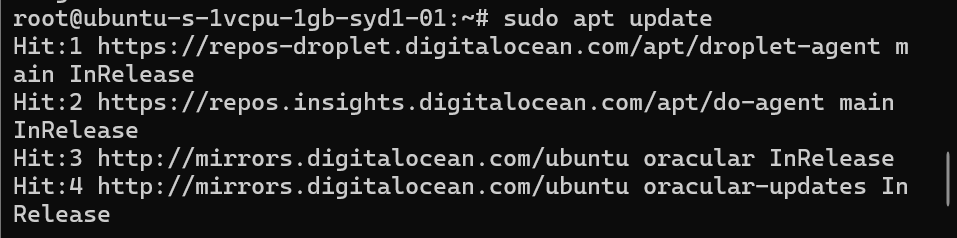
## **6. Add Universe Repository**

One of the first weird things I had to do was enable this “universe” repository on Ubuntu. Apparently, the regular setup doesn’t give you every program you might need, so you have to unlock more stuff yourself. This step is pretty important, especially for things like Certbot that you need later for HTTPS.

* Command I used: sudo add-apt-repository universe
* Then I did: sudo apt update
* After this, my server could find and install a bunch of extra software I wouldn’t get otherwise.



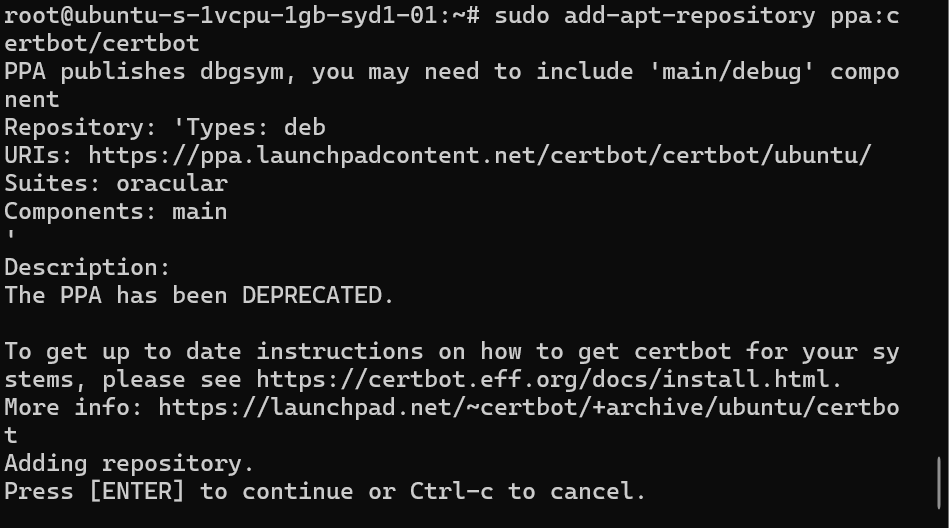
sudo apt update



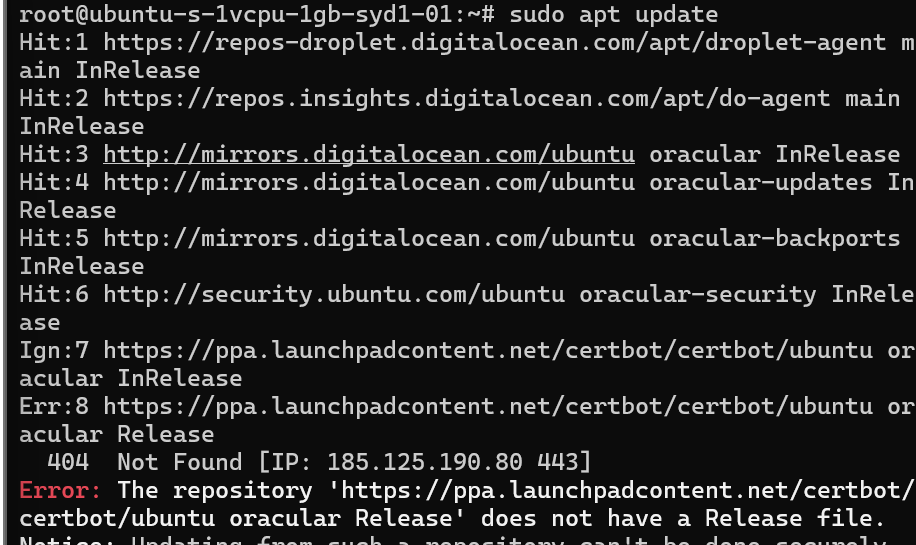
## **7. Add Certbot PPA**

Certbot is what lets you get those free SSL certificates so your site is secure and has that padlock. But the version in Ubuntu can be pretty old, so I had to add a special source (they call it a PPA) to make sure I’d get the newest Certbot. Without this, you can’t really keep Certbot updated properly, which could mess up your HTTPS later on.

* I ran: sudo add-apt-repository ppa:certbot/certbot
* And then: sudo apt update
* Doing this meant I could install Certbot without any issues and keep it working for the future.



sudo apt update

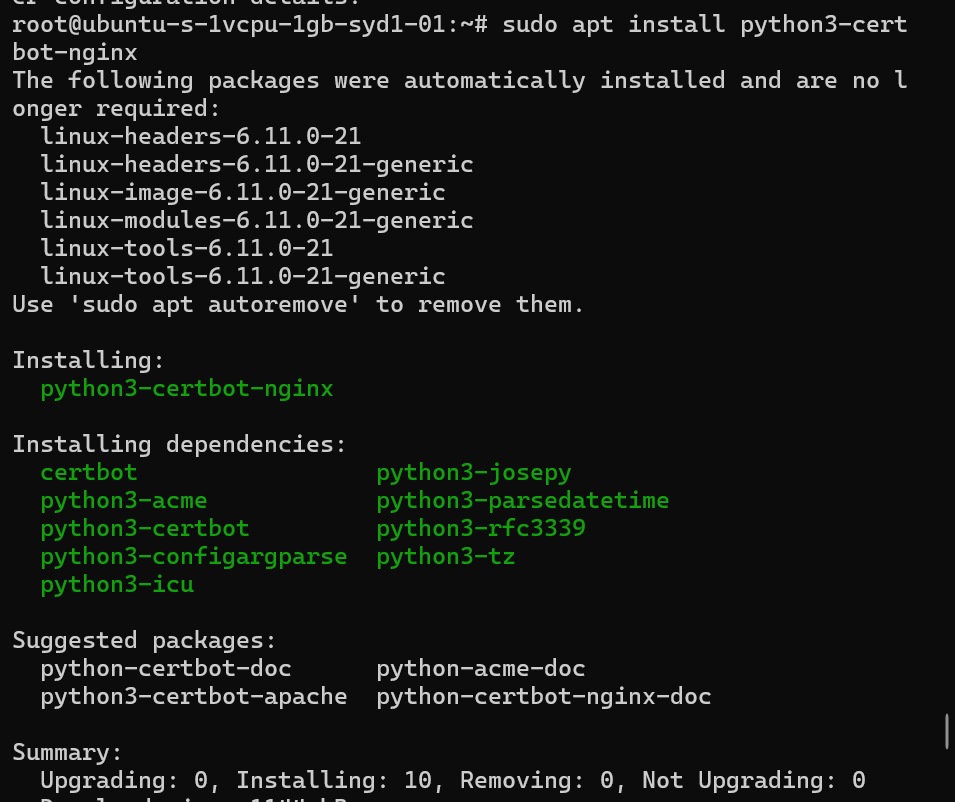


## **8. Install Certbot’s Nginx Package**

To make Certbot actually work with my Nginx server, I had to install its special Nginx package. Without this, Certbot doesn’t really know how to set up HTTPS on Nginx automatically. After running this, it just kind of connects the two and makes it way easier to set up SSL or renew certificates.

* I used: sudo apt install python3-certbot-nginx
* This made it so Certbot could talk to Nginx and do all the HTTPS setup for me, which saved a ton of time.

sudo apt install python3-certbot-nginx



## **9. Setup SSL/TLS with Certbot for Nginx**

Getting SSL/TLS working is super important these days so people’s info is safe and your site shows that padlock. I used Certbot with Nginx to grab a free SSL cert from Let’s Encrypt and set everything up. The command was actually pretty simple, and Certbot did all the tricky stuff for me.

* The main command: sudo certbot --nginx -d anuj.fun
* Certbot went and fixed my Nginx config, so now anyone who goes to http just gets sent straight to https.
* It even set up the certs to renew automatically, so I don’t have to stress about it expiring. This not only makes the site secure but also looks more legit in Google and for anyone visiting.

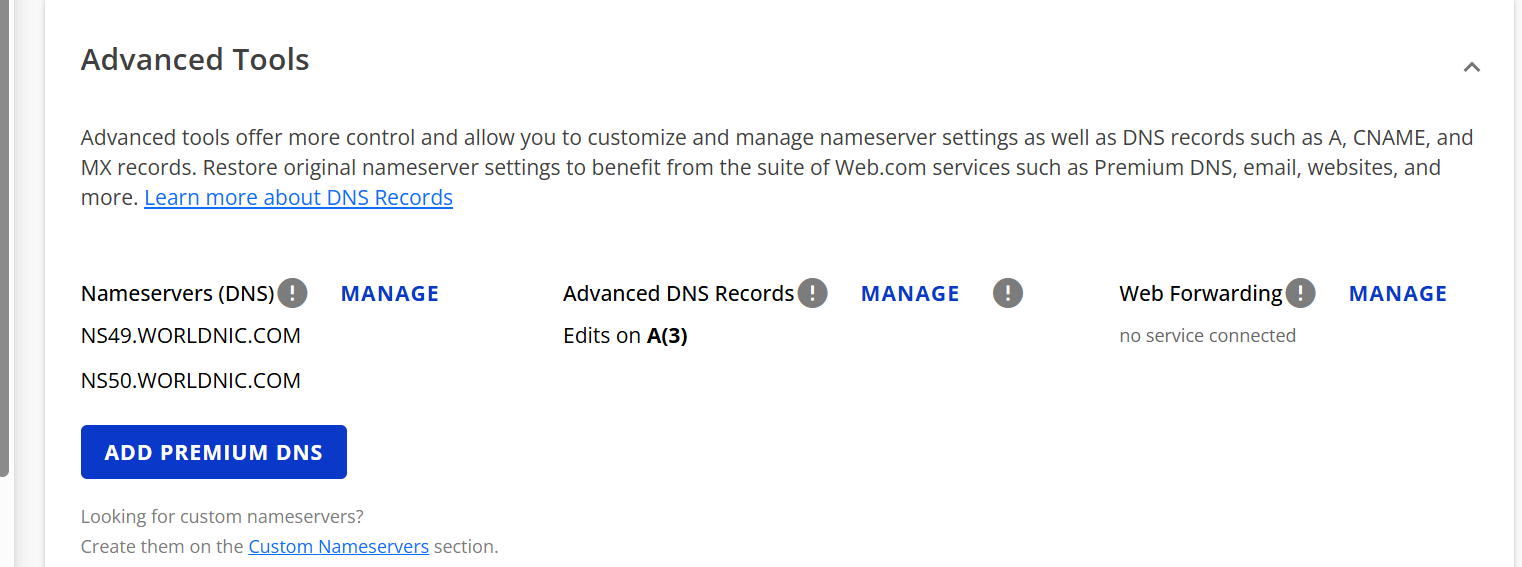
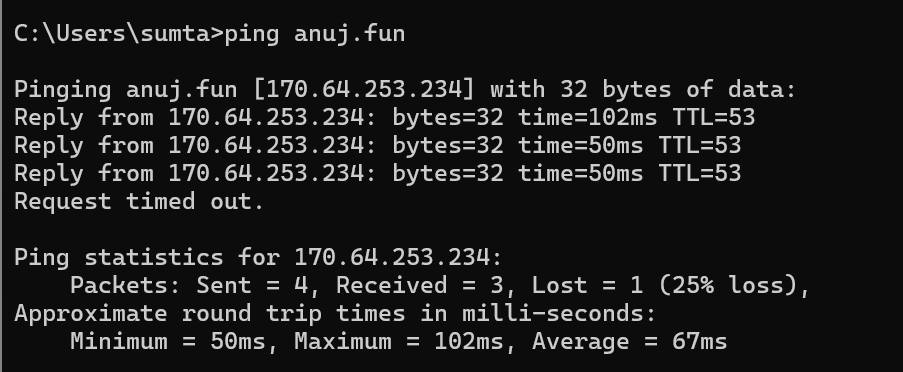
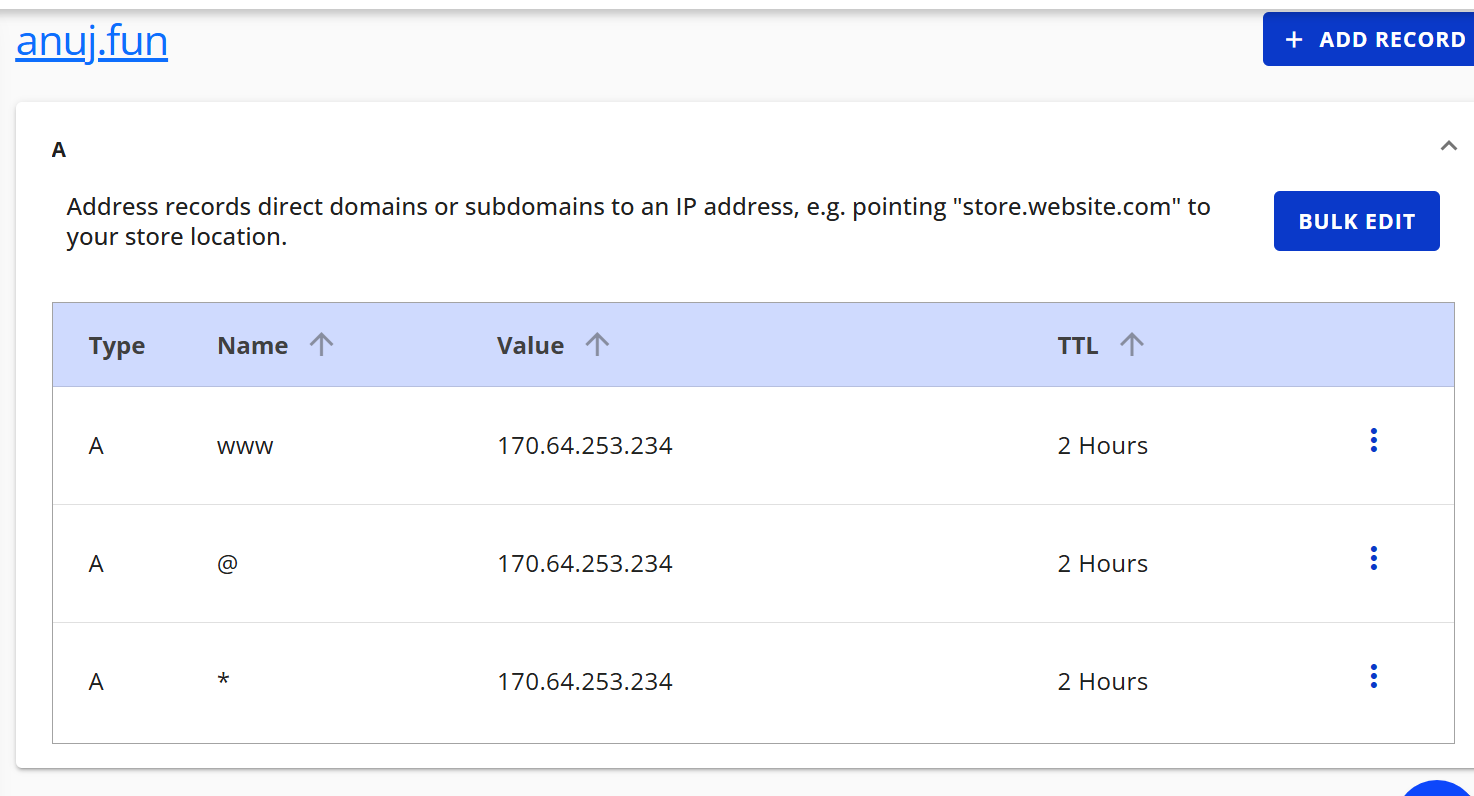
sudo certbot --nginx -d anuj.fun



**Screenshot**: Successful Certbot output, browser showing secure HTTPS.

## **10. DNS Functionality and Accessibility**

For the domain part, I had to go into where I bought my domain(web.com) and set up what they call an “A” record. Basically, I pointed “anuj.fun” at my server’s IP (170.64.253.234). It didn’t work right away—sometimes DNS just takes a while to catch up, which was kind of annoying.

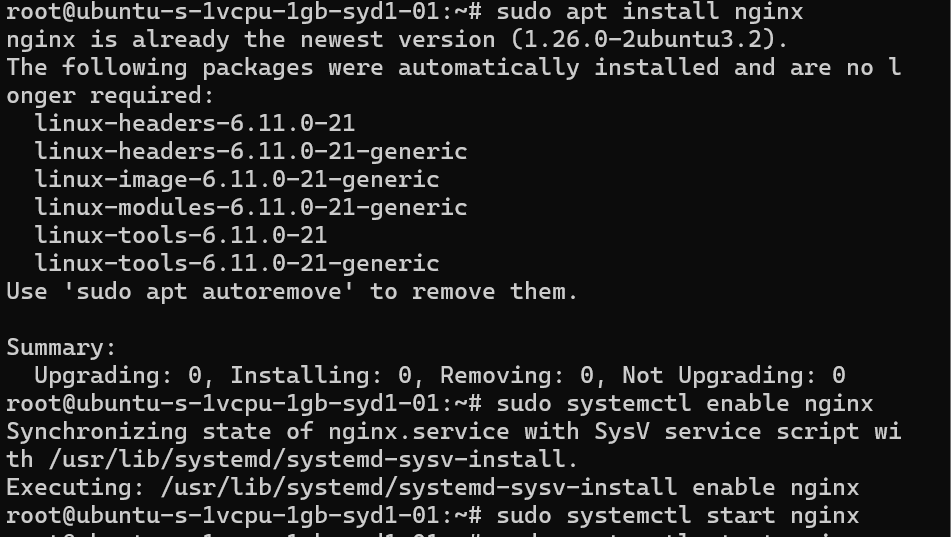
* I kept checking with nslookup and ping in the terminal, plus used some online DNS checker sites.
* After a bit of waiting, my domain finally started sending people to my server, and I knew it was all set up right.
* DNS Provider settings verification
* DNS propagation check  
   **Screenshot**: Verified DNS resolution with ping and DNS lookup.
* 

## **11. Web Server Configuration (Nginx)**

I set up Nginx as my main web server to run the website. The install was pretty easy—just a couple commands—and I made sure it always started up if the server rebooted. Didn’t really change much in the config files at first, just wanted it to work for both http and https.

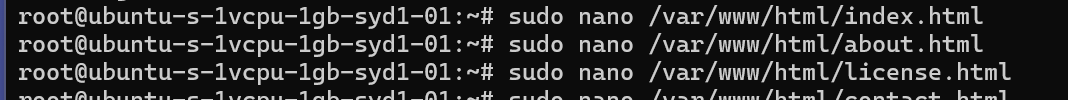
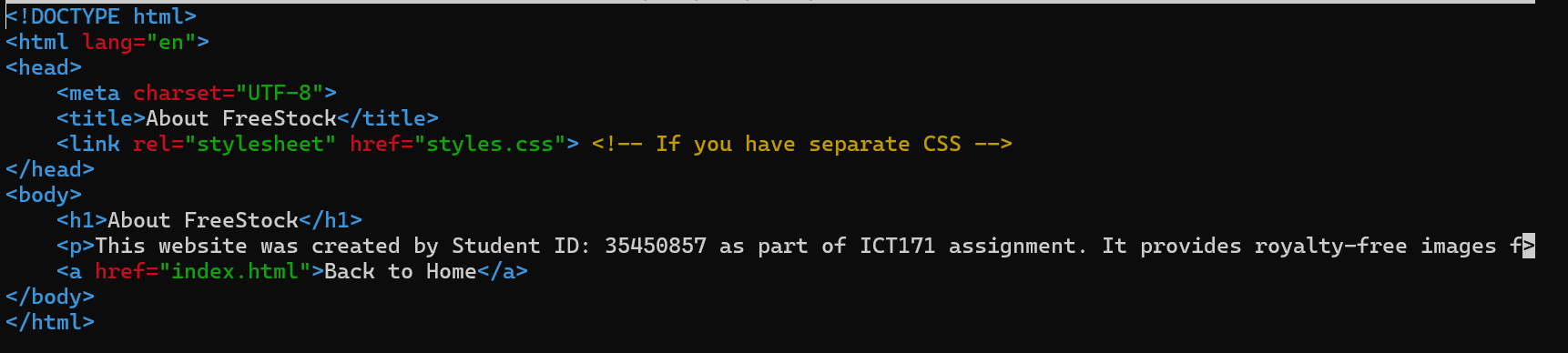
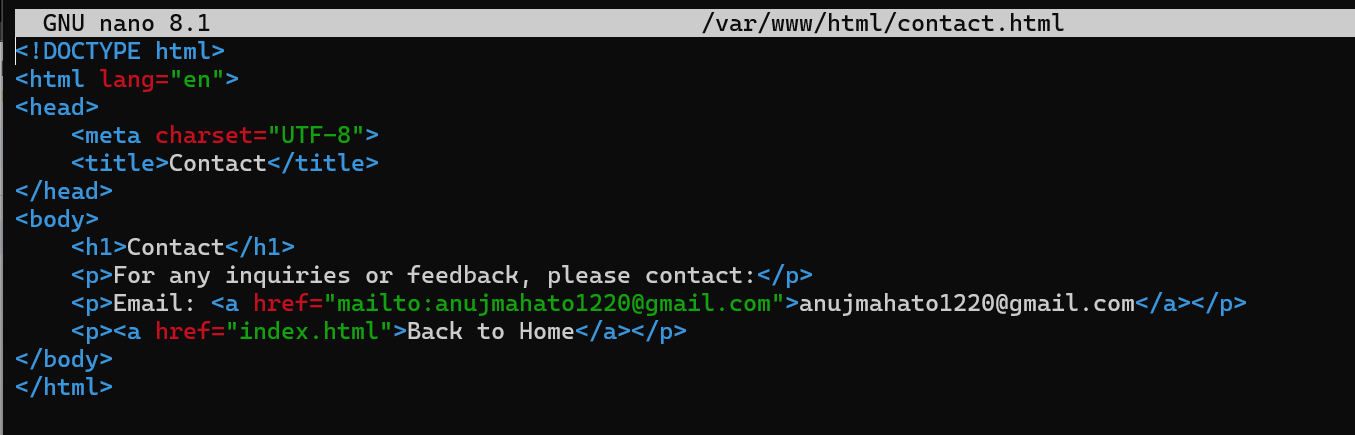
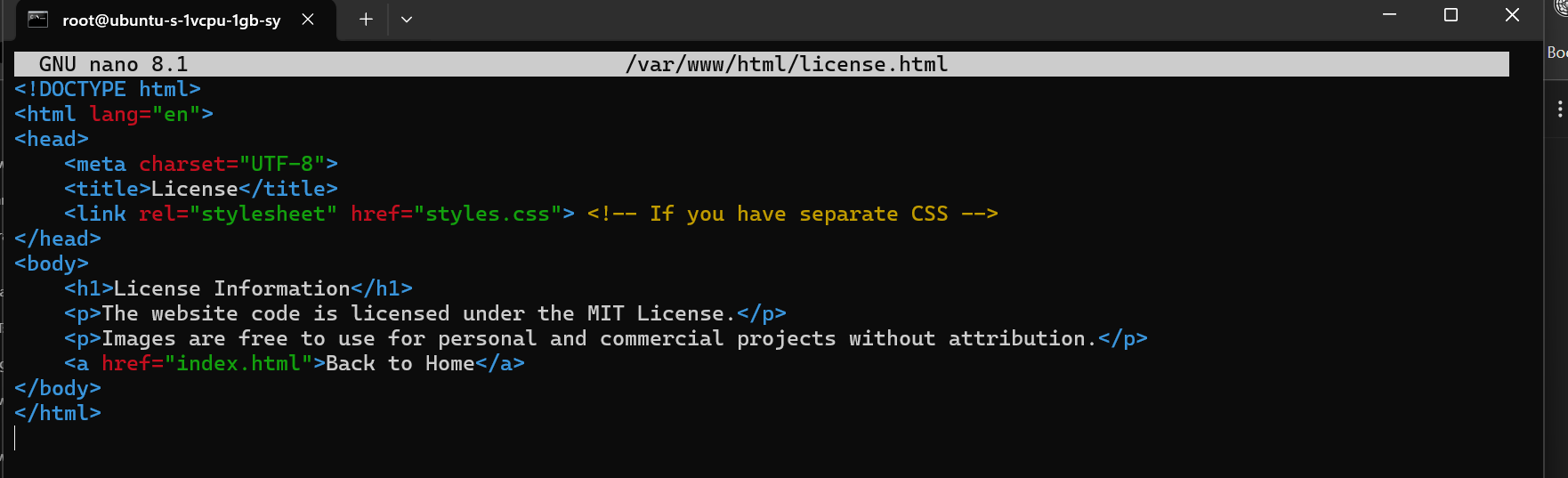
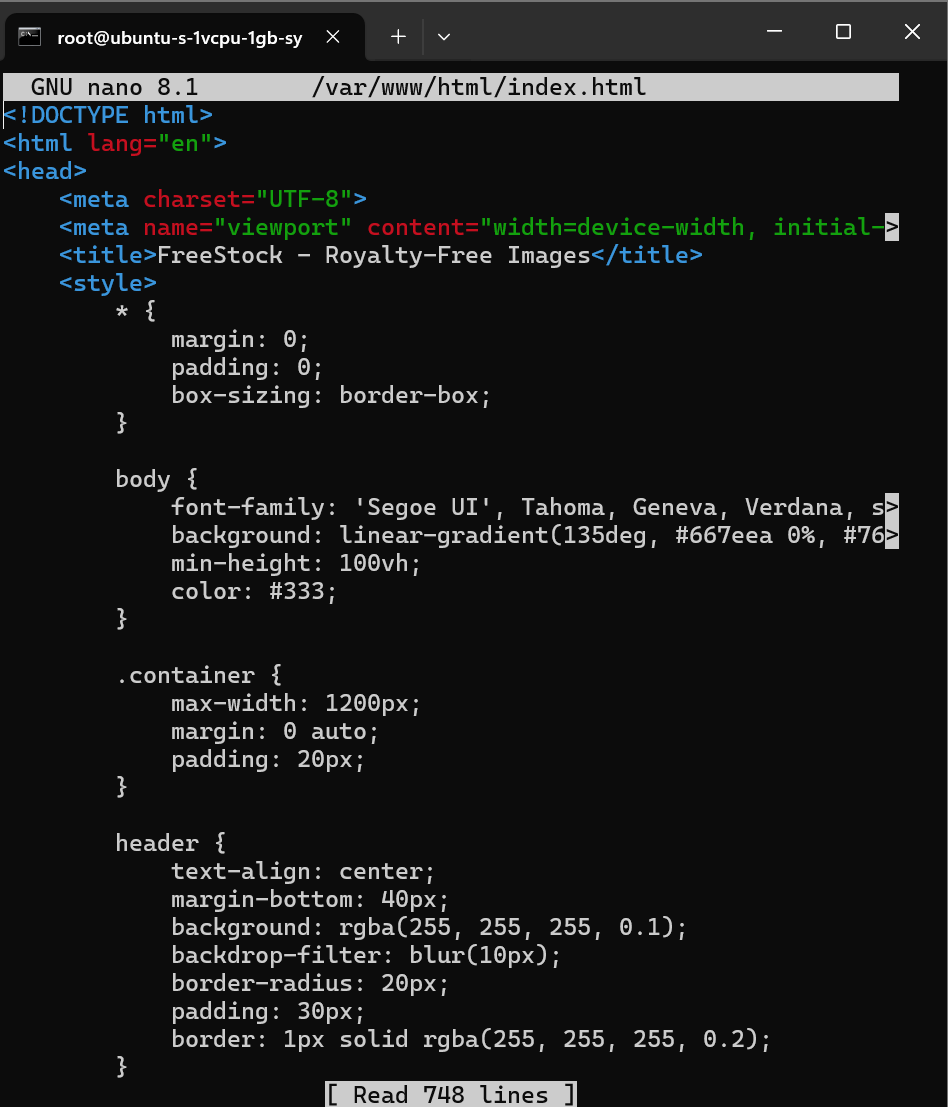
* Commands I used:  
  + sudo apt install nginx
  + sudo systemctl enable nginx
  + sudo systemctl start nginx
* To check it all worked, I just went to my domain and saw the Nginx welcome page. Later I swapped that out for my own index.html with my project info. After that, my site was running and ready for people to visit.

**Screenshot**: Nginx configuration and running status.



## **12. Editing HTML Page**

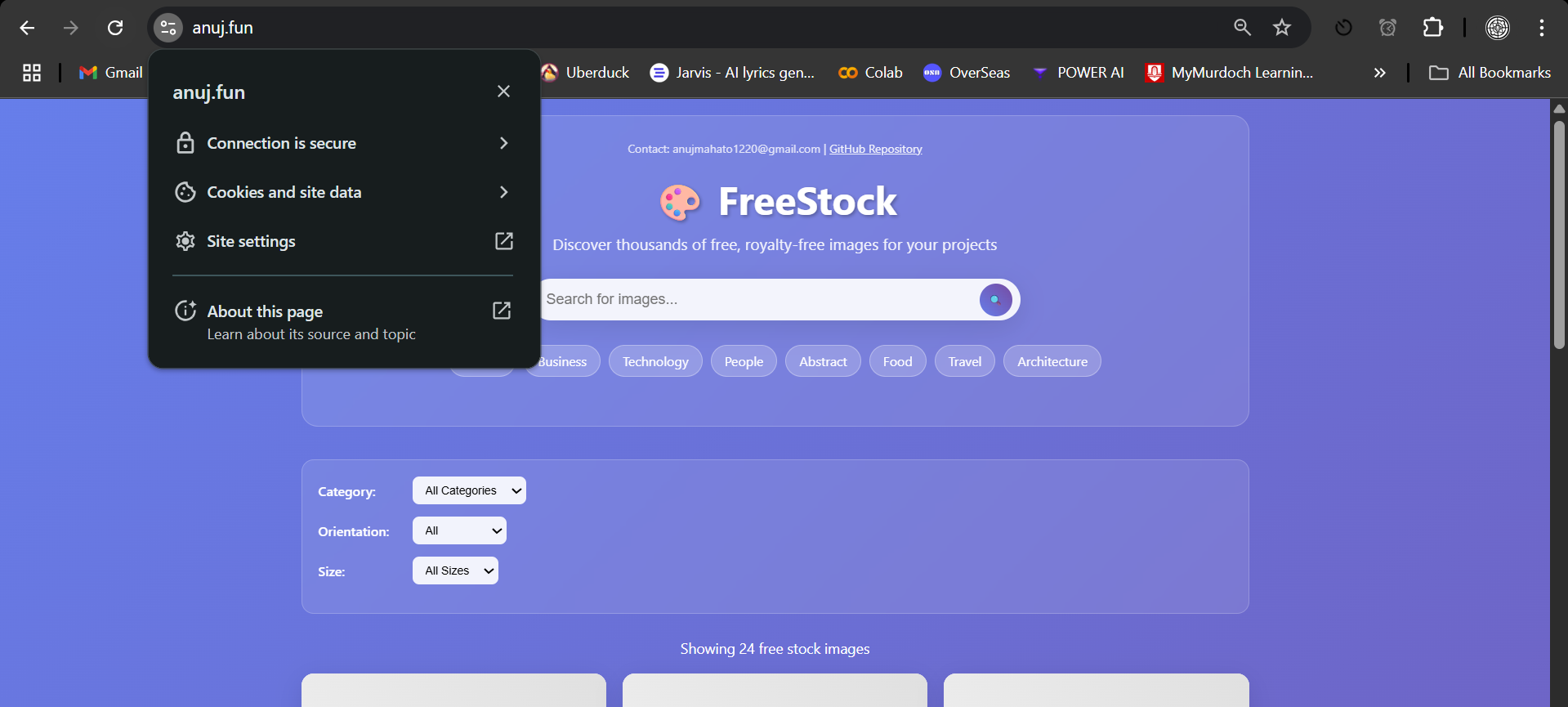
Customized default HTML content to include project description, student ID, and essential information:

* File path: /var/www/html/index.html
* HTML structure explained and modified  
   **Screenshot**: Code snippet showing edits.

## **13. Final Website and Functionality**

The finished website is up and running at<https://anuj.fun> for anyone to check out. The home page has a short welcome, my project info, my student ID, and I made sure to put the MIT license down the bottom so it’s clear anyone can use the code. All those setup steps finally led to a proper website that’s live with https (and the padlock shows it’s secure) on my own domain name. I even tested it on my phone and some friends’ laptops just to make sure it loads everywhere and looks right.

Screenshot: Just take a screenshot of the homepage in a browser, showing the domain, padlock, and all the info on the page.



## **14. GitHub Repository Submission & Commits**

I put all my code, scripts, and files in a public GitHub repo so it’s easy for anyone to see what I did or even copy it if they want. Tried to remember to make a commit after every big change, like when I set stuff up, changed the html, or worked on the scripts. Each commit has a short note so I remember what I changed. My repo’s got everything: index.html, server.sh, backup\_script.sh, plus a README that explains how it works and how to set it up yourself.

[GitHub Link:](https://github.com/AJ-1220/ANUJ-ICT171) **Screenshot:** Screenshot of the repo’s commit history and file structure.

## **15. Bash Scripting (server.sh)**

Automated routine server setup tasks with a clearly commented script:

To automate server setup, I wrote a bash script (server.sh) that updates system packages, installs and configures Nginx, and prepares the environment for further deployment. This script reduces manual steps and increases repeatability. Additionally, I created a backup script that compresses the website’s documents into a date-stamped archive, safeguarding project data.

Both scripts are thoroughly commented to explain each command and are included in the GitHub repo for review. Testing and verification confirmed that running the scripts resulted in a working server and reliable automated backups.

#!/bin/bash

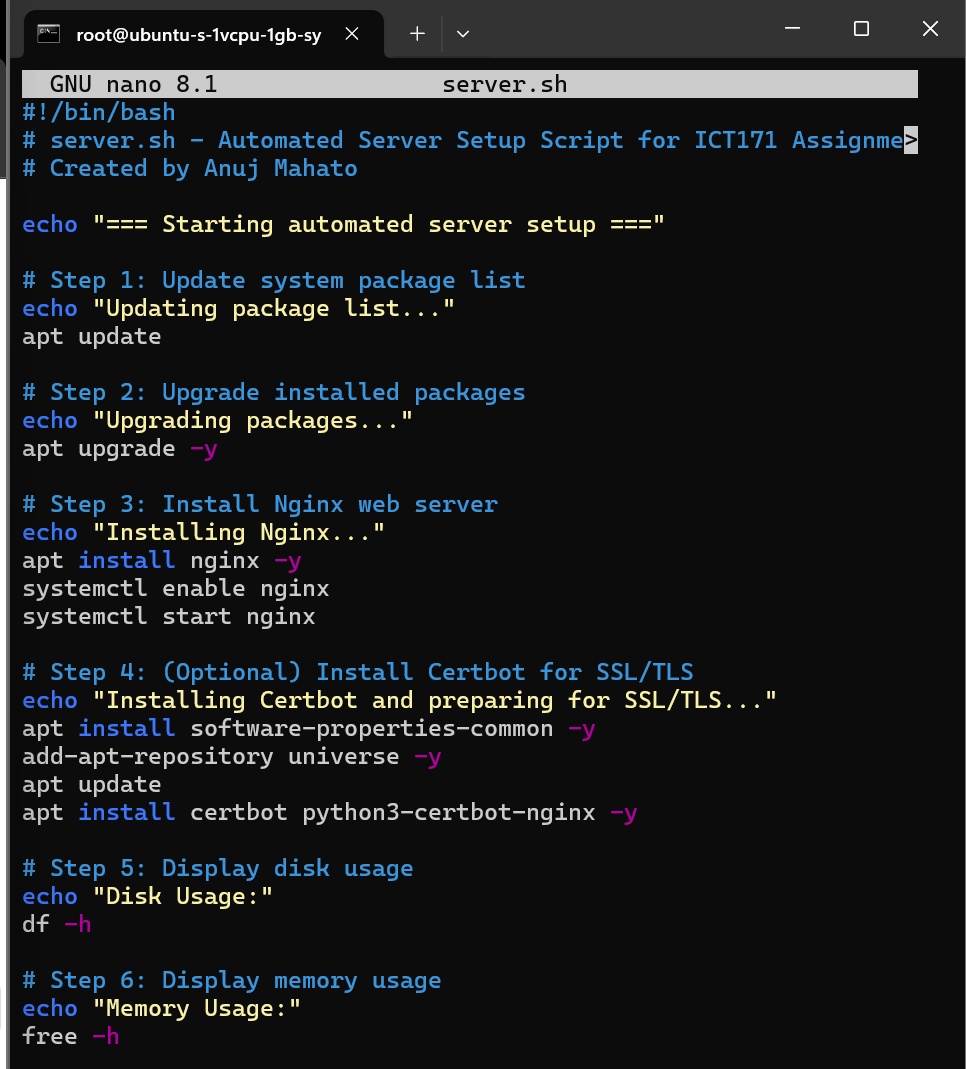
# Updates and installs necessary software

sudo apt update

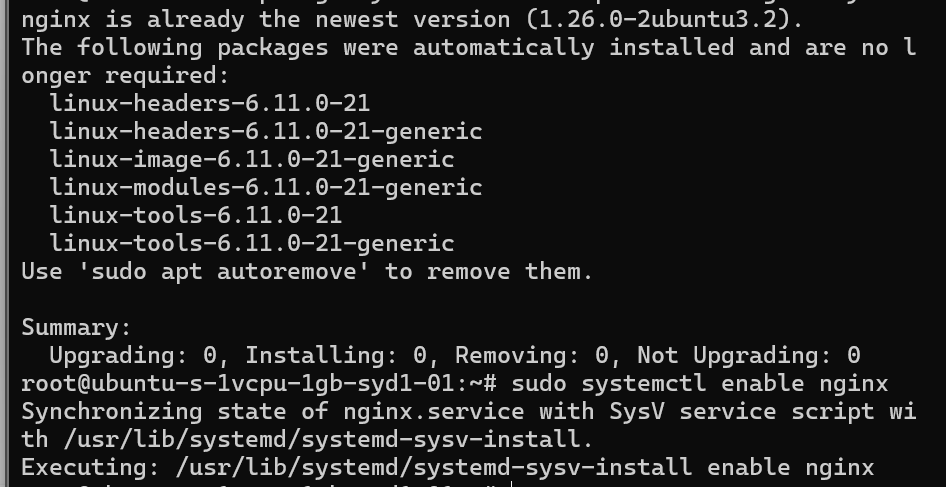
sudo apt upgrade -y

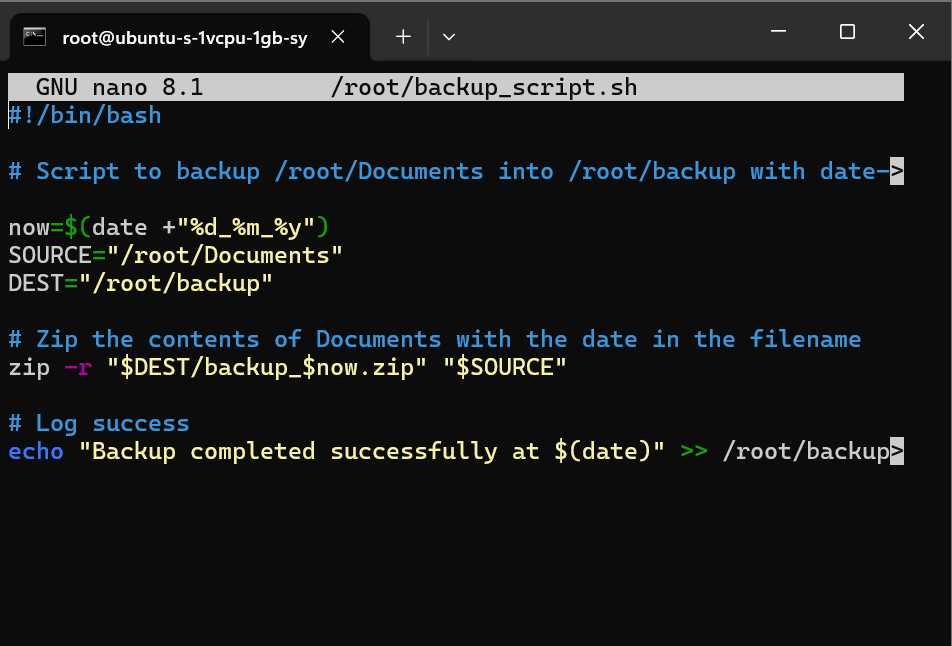
sudo apt install nginx -y

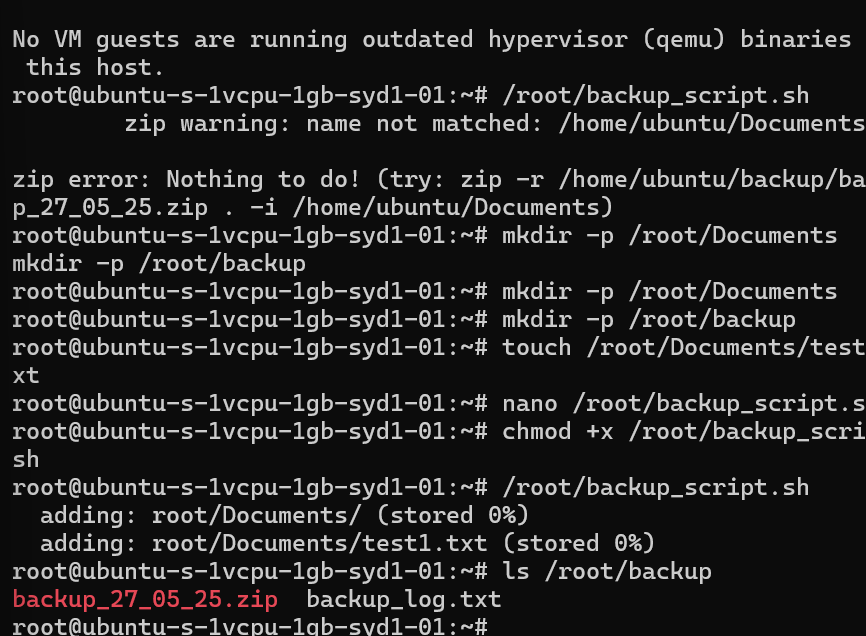
sudo systemctl enable nginx

**Screenshot**: Execution of server.sh in terminal.

For [Backup.sh](http://backup.sh)







## **16. Windows Scripting (File Transfer & Server Access)**

I needed to move files between my Windows laptop and my cloud server, so I used Command Prompt and sometimes PowerShell to connect to the server. I mostly used the scp command (or WinSCP if I wanted to just drag and drop) to send files like index.html straight into the web folder on my server.

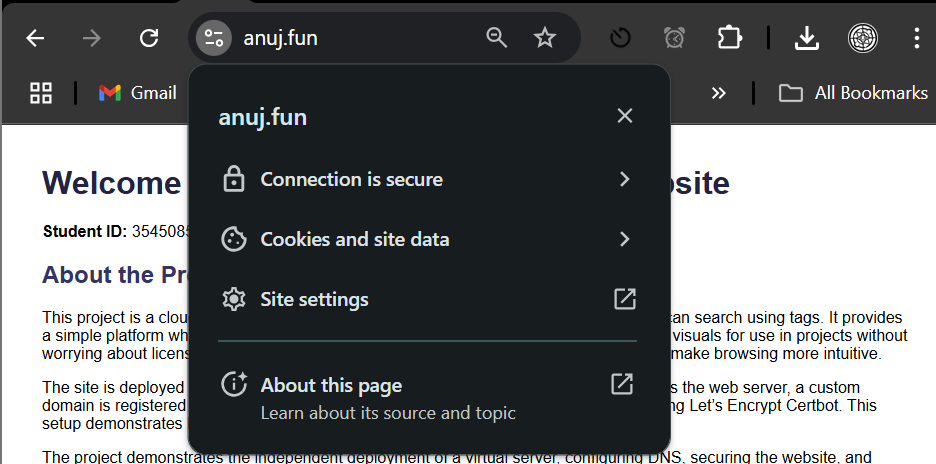
After putting in my server password, I could upload or even download files between Windows and Linux without any hassle. Doing this showed I can handle remote server stuff and keep my project up to date from my own computer, not just in the Linux terminal.

## **17. Script Usefulness and Output**

Making scripts for setting up the server and backups honestly saves heaps of time. Instead of typing out all the steps every single time, I just run the script and it does everything for me in a few seconds.

* Less chance for mistakes too, since the script always does it the same way every time.
* If I (or someone else) ever needs to rebuild the site or get the files back, they just run the script and it works—no special tricks needed.
* Right after the scripts finish, my website is up and backups are sitting there in the folder.

I checked everything after running them, and both the site and backup files were exactly where they should be, so I know the scripts worked.

**Screenshot**: Resulting site functionality after running scripts.

## **18. License Explanation**

I picked the MIT License for my project ’cause it’s simple and super open. Anyone can use my code, change it up, or share it, as long as they give me credit and keep the license bit in there. It just fits with the point of the project—making it easy for other students or anyone to use what I made without worrying about legal stuff. So, anyone can build on this for their own uni work or just for learning.

The full license text is available in both the website footer and the GitHub repository.

## **19. Video Demonstration & Script**

A concise, engaging video clearly demonstrates:

https://youtu.be/HqxH3ka98Qo

* Project introduction and goals
* Live demonstration of website features and secure connection
* Explanation and demonstration of SSH and scripting automation
* GitHub repository review
* Concluding statements  
   **Script**: Clearly written and aligned with spoken presentation.

### **References**

1. **DigitalOcean documentation:** https://docs.digitalocean.com/
2. **Certbot user guide:** https://certbot.eff.org/docs/
3. **Nginx official documentation:** https://nginx.org/en/docs/