SSW 590 Group 11 Version: GIT_HASH_HERE

by

Charles Villa, Justin Phan, Benedict Martinez, Jacky Lei

cvilla@stevens.edu, jphan1@stevens.edu, bmartin5@stevens.edu, jlei7@stevens.edu

October 22, 2025

cvilla@stevens.edu, jphan1@stevens.edu, bmartin5@stevens.edu, jlei7@stevens.edu

SSW 590 Group 11 Version: GIT_HASH_HERE

Charles Villa, Justin Phan, Benedict Martinez, Jacky Lei
cvilla@stevens.edu, jphan1@stevens.edu, bmartin5@stevens.edu, jlei7@stevens.edu

This document provides the requirements and design details of the PROJECT. The following table (Table 1) should be updated by authors whenever major changes are made to the architecture design or new components are added. Add updates to the top of the table. Most recent changes to the document should be seen first and the oldest last.

Table 1: Document Update History

Date	**Updates**
10/17/2025	JP, BM, JL, CV: • Added section detailing using Caddy for SSL certificates. 9)
10/16/2025	JP, BM, JL, CV: • Added host and password for Overleaf container. 1) 2)
10/15/2025	JP, BM, JL, CV: • Created Domain Names, SSL, and Versioning chapter 9)
10/8/2025	JP, BM, JL, CV: • Finalized Passwords table with a consistent algorithm. Linked each entry to Hosts. Added Overleaf Docker host row. 1) 2)
10/7/2025	JP, BM, JL, CV: • Created Overleaf chapter 8)
10/7/2025	JP, BM, JL, CV: • Updated Hosts 1.1 and Passwords tables 2.1
10/6/2025	JP, BM, JL, CV: • Created Bugzilla chapter 7)
09/24/2025	JP, BM, JL, CV: • Created LaTeX Docker chapter (Chapter 6)
09/24/2025	JP, BM, JL, CV: • Created AWS Deployment chapter (Chapter 5)
09/24/2025	JP, BM, JL, CV: • Created Project Proposal chapter (Chapter 4)
09/17/2025	BM: • Finalized Parts I, J, and H

Table 1: Document Update History

Date	**Updates**
09/16/2025	JP: • Created sections Part G and Part I
09/16/2025	CV: • Finalized Parts C, D, E
09/15/2025	JL: • Created Parts A and B

Table of Contents

List of Tables

List of Figures

Chapter 1

Hosts

– Charles, Justin, Benedict, Jacky

Table 1.1: Hosts Table (Edited after Bugzilla Assignment)

Name	**IP Address (or IP:Port)**	**OS**	**Job**
devbox	10.0.0.10	Windows	Primary workstation used for coding and pushing commits to GitHub repositories.
database	10.0.0.11	Linux	Stores all persistent project data and connects to the backend API host.
testing	10.0.0.12	Linux	Dedicated environment for integration and regression testing prior to deployment.
api	10.0.0.13	Ubuntu 22.04	Backend REST API server responsible for serving requests between frontend and database.
Bugzilla Docker	174.138.69.132:8080	Ubuntu	Docker container hosting Bugzilla (public HTTP access via port 8080).
Overleaf Docker	104.236.74.225:80	Ubuntu	Overleaf Community Edition instance (public HTTP access on port 80).

Chapter 2

Passwords

– Charles, Justin, Benedict, Jacky

Table 2.1: Password Table

User / Account	Password (Hint)	Server Rules / Notes
bugzilla_admin	<KEY><N>@bugzilla	For the Bugzilla Docker container (174.138.69.132:8080). Follows the shared key rule format.
overleaf_maintainer	<KEY><N>@overleaf	For the Overleaf Docker container (104.236.74.225:80). Same structure as others for maintainability.
api_svc	<KEY><N>@api	Used by the backend API host (10.0.0.13).
db_admin	<KEY><N>@db	Used for the main database host (10.0.0.11).
devbox_admin	<KEY><N>@devbox	Used for the development workstation (10.0.0.10).
tester	<KEY><N>@testing	Used for the testing host (10.0.0.12).

Each password follows our shared group convention: a short English word beginning and ending with the same letter (<KEY>), followed by a number equal to twice the number of vowels in that word (<N>), and ending with the site tag. This system allows easy password rotation while keeping host associations clear and consistent.

Chapter 3

Linux Commands

– Charles, Justin, Benedict, Jacky

3.1 Part A: Navigation & File Ops

- 1:

```
(base) jackylei@Jackys-MacBook-Pro lx-test % pwd
/Users/jackylei/lx-test

```

- 2:

```
(base) jackylei@Jackys-MacBook-Pro lx-test % ls -a -lh
total 136
drwxr-xr-x@ 12 jackylei  staff   384B Sep 15 16:56 .
drwxr-x---+ 58 jackylei  staff   1.8K Sep 13 22:14 ..
-rw-r--r--@  1 jackylei  staff   6.0K Sep 15 16:53 .DS_Store
drwxr-xr-x@  3 jackylei  staff    96B Sep 15 16:52 archive
-rw-r--r--@  1 jackylei  staff    48K Sep 13 22:14 blob.bin
lrwxr-xr-x@  1 jackylei  staff    13B Sep 13 22:14 link-to-file1 -> src/
    file1.txt
-rw-r--r--@  1 jackylei  staff     0B Sep 15 16:56 notes.md
-rw-r--r--@  1 jackylei  staff    56B Sep 15 16:56 people.csv
drwxr-xr-x@  5 jackylei  staff   160B Sep 13 22:14 src
-rw-r--r--@  1 jackylei  staff    56B Sep 13 22:14 sys.log
drwxr-xr-x@  3 jackylei  staff    96B Sep 15 16:04 tmp
-rw-r--r--@  1 jackylei  staff    28B Sep 13 22:14 words.txt

```

- 3:

```
(base) jackylei@Jackys-MacBook-Pro lx-test % [ -d tmp ] && cp -v src/
    file1.txt tmp
src/file1.txt -> tmp/file1.txt

```

- 4:

```
(base) jackylei@Jackys-MacBook-Pro lx-test % mv -v old.txt archive/
old.txt -> archive/old.txt

```

- 5:

```
(base) jackylei@Jackys-MacBook-Pro lx-test % touch notes.md

```

- 6:

```
(base) jackylei@Jackys-MacBook-Pro lx-test % du -h src

```

3.2 Part B: Viewing & Searching

- 7:

```
(base) jackylei@Jackys-MacBook-Pro lx-test % cat -n sys.log
     1  INFO boot ok
     2  WARN disk low
     3  ERROR fan fail
     4  INFO shutdown

```

- 8:

```
(base) jackylei@Jackys-MacBook-Pro lx-test % cat sys.log | grep "ERROR"
ERROR fan fail

```

- 9:

```
(base) jackylei@Jackys-MacBook-Pro lx-test % grep -o -i '[[:alnum:]]\+'
    words.txt | sort -u | wc -l
       4

```

- 10:

```
(base) jackylei@Jackys-MacBook-Pro lx-test % cat words.txt | grep -i "g"
Gamma
gamma

```

- 11:

```
(base) jackylei@Jackys-MacBook-Pro lx-test % head -2 people.csv
id,name,dept
1,Ada,EE

```

- 12:

```
(base) jackylei@Jackys-MacBook-Pro lx-test % tail -3 sys.log
WARN disk low
ERROR fan fail
INFO shutdown

```

3.3 Part C: Text Processing

- 13:

```
awk -F ',' 'NR > 1 {print $2}' people.csv

```

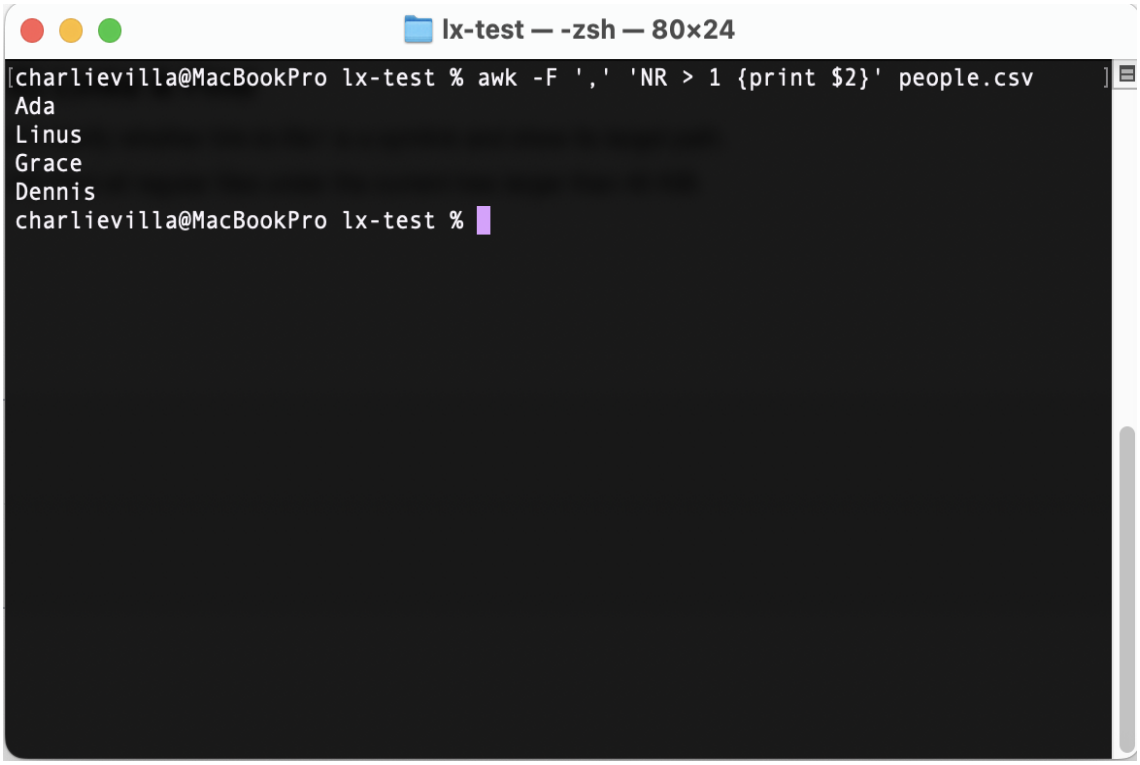

Figure 3.1: Part C #13 Terminal Output

- 14:

```
sort -f -u words.txt

```

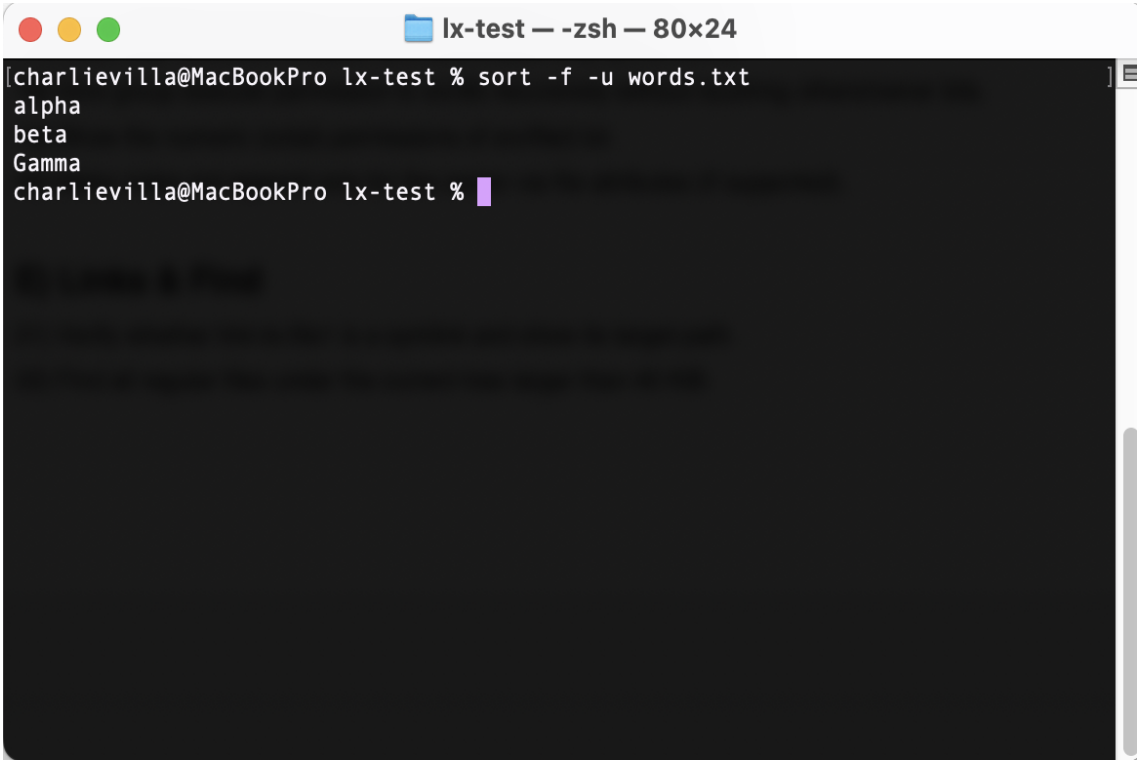

Figure 3.2: Part C #14 Terminal Output

- 15:

```
find src/ -type f -exec sed -i.bak 's/three/3/g' {} +
```

No terminal output for question 15

- 16:

```
wc src/*.txt
```

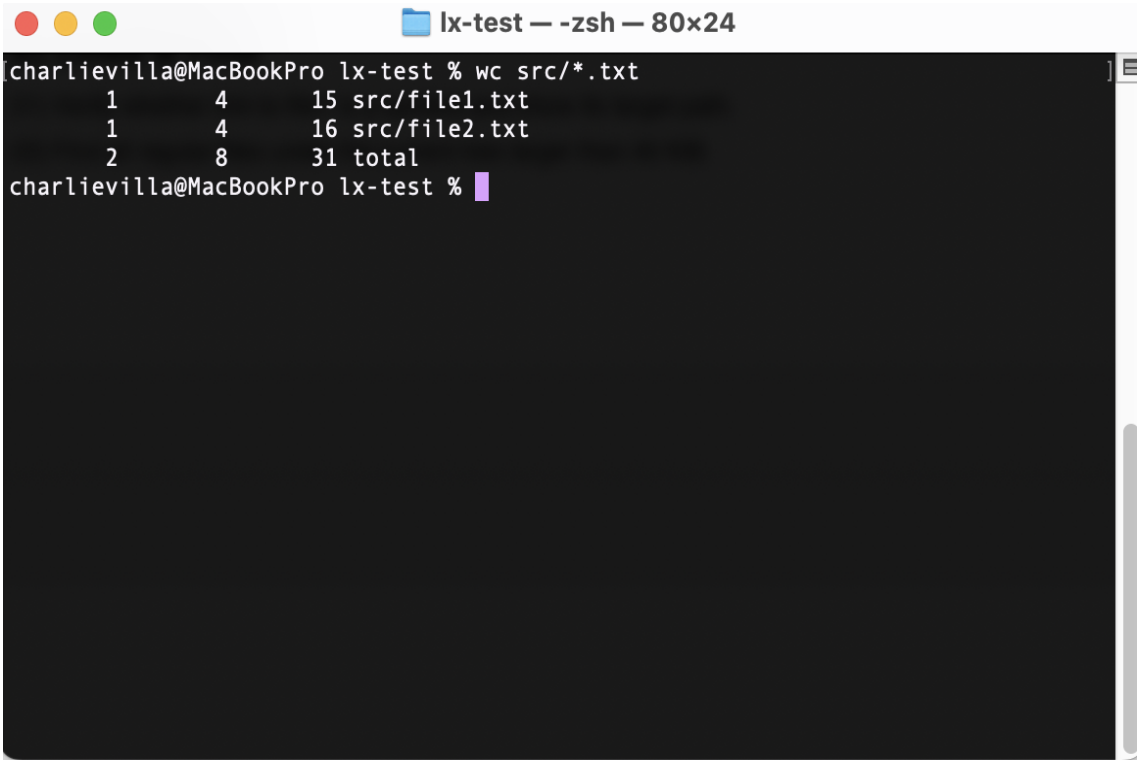

Figure 3.3: Part C #16 Terminal Output

3.4 Part D: Permissions & Ownership

- 17:

```
chmod 700 tmp/

```

No terminal output for question 17

- 18:

```
chmod -R g+x src/lib

```

No terminal output for question 18

- 19:

```
stat -f %p src/file2.txt

```

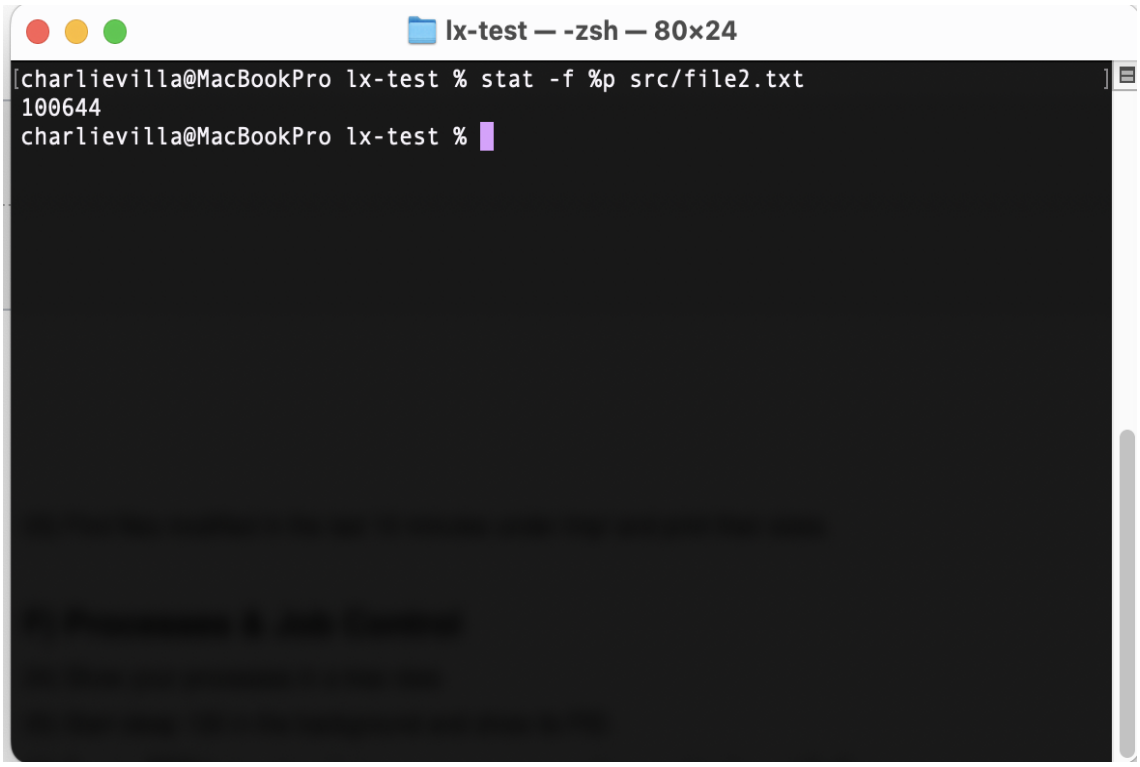

Figure 3.4: Part C #19 Terminal Output

- 20:

```
touch notes.md
chflags uappnd notes.md

```

No terminal output for question 20

3.5 Part E: Links & Find

- 21:

```
ls -l link-to-file1

```

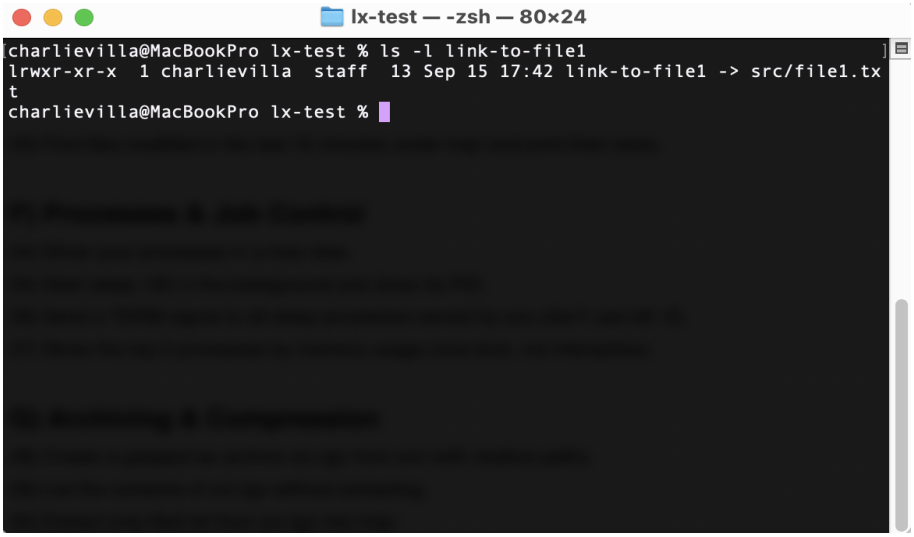

Figure 3.5: Part C #21 Terminal Output

- 22:

```
find . -type f -size +40k

```

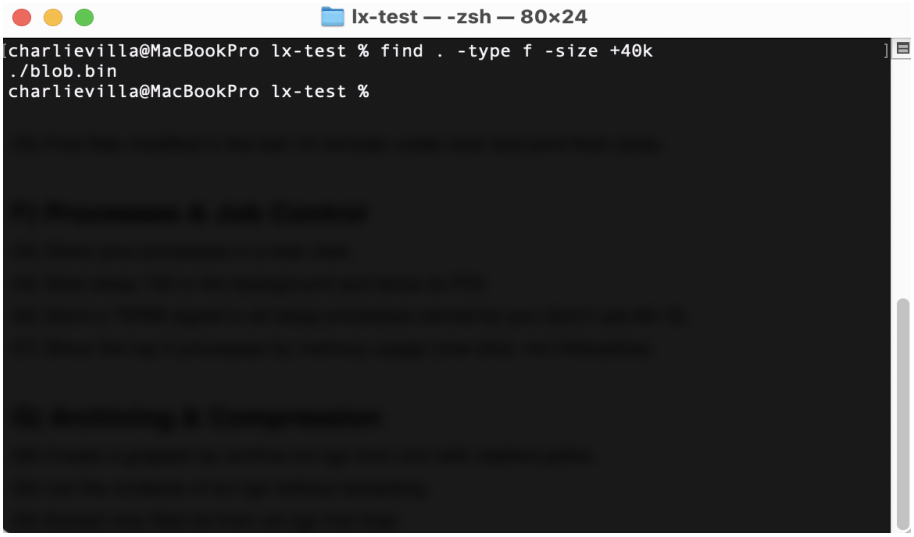

Figure 3.6: Part C #22 Terminal Output

- 23:

```
touch tmp/some-new-file.txt
find tmp/ -type f -mmin -10 -exec stat -f "%z %N" {} +

```

Created a new file with "touch" because the tmp/ directory was empty before it.

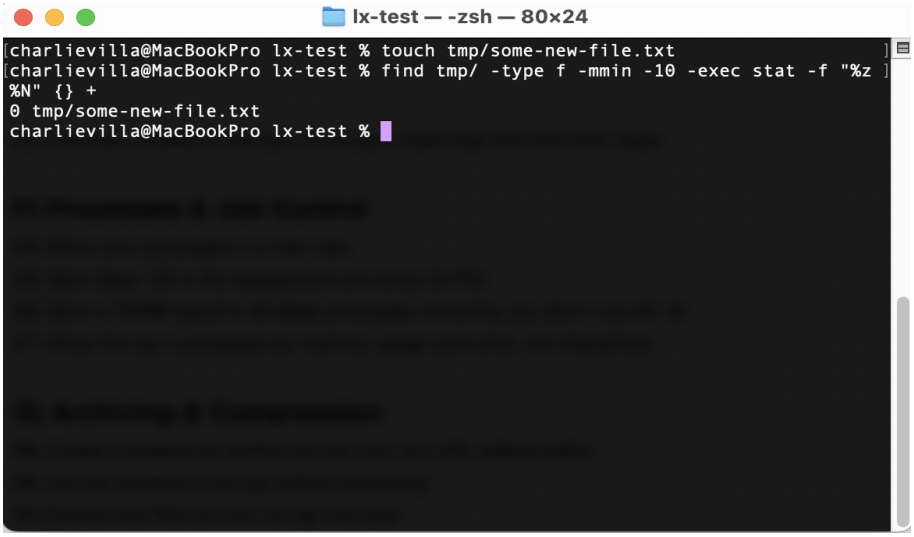

Figure 3.7: Part C #23 Terminal Output

3.6 Part F: Processes & Job Control

- 24: Tree View:

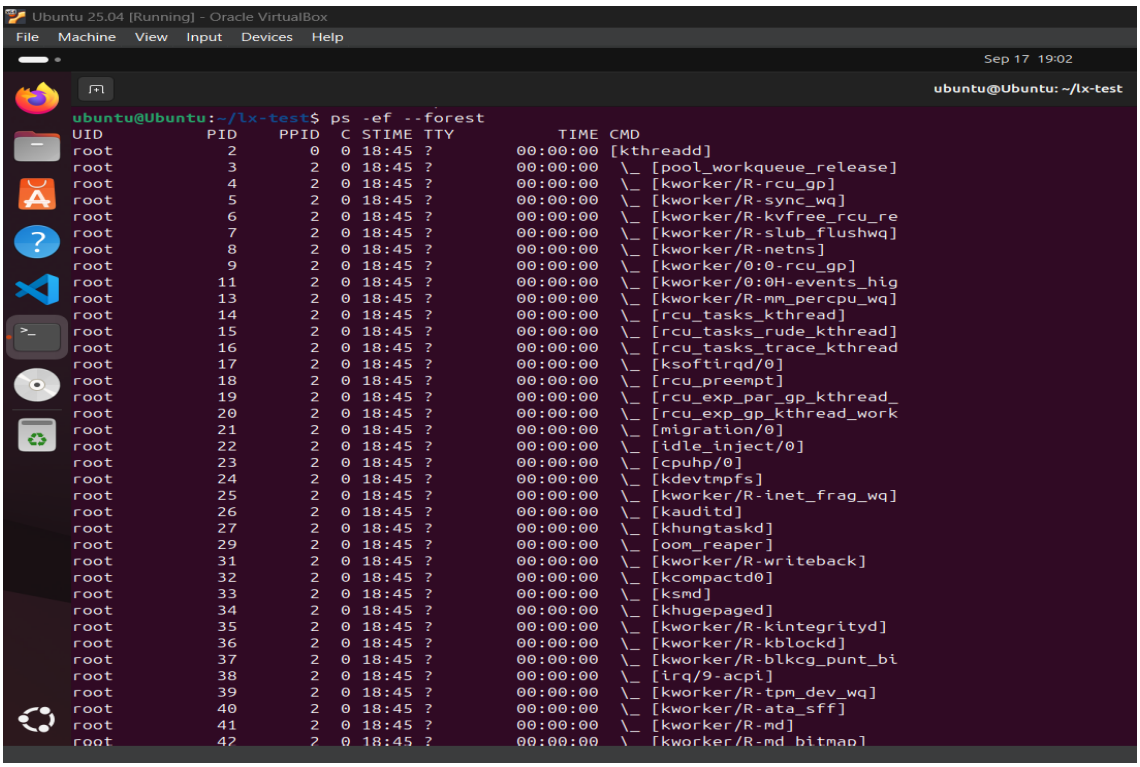

Figure 3.8: Part F #24 Tree View 1

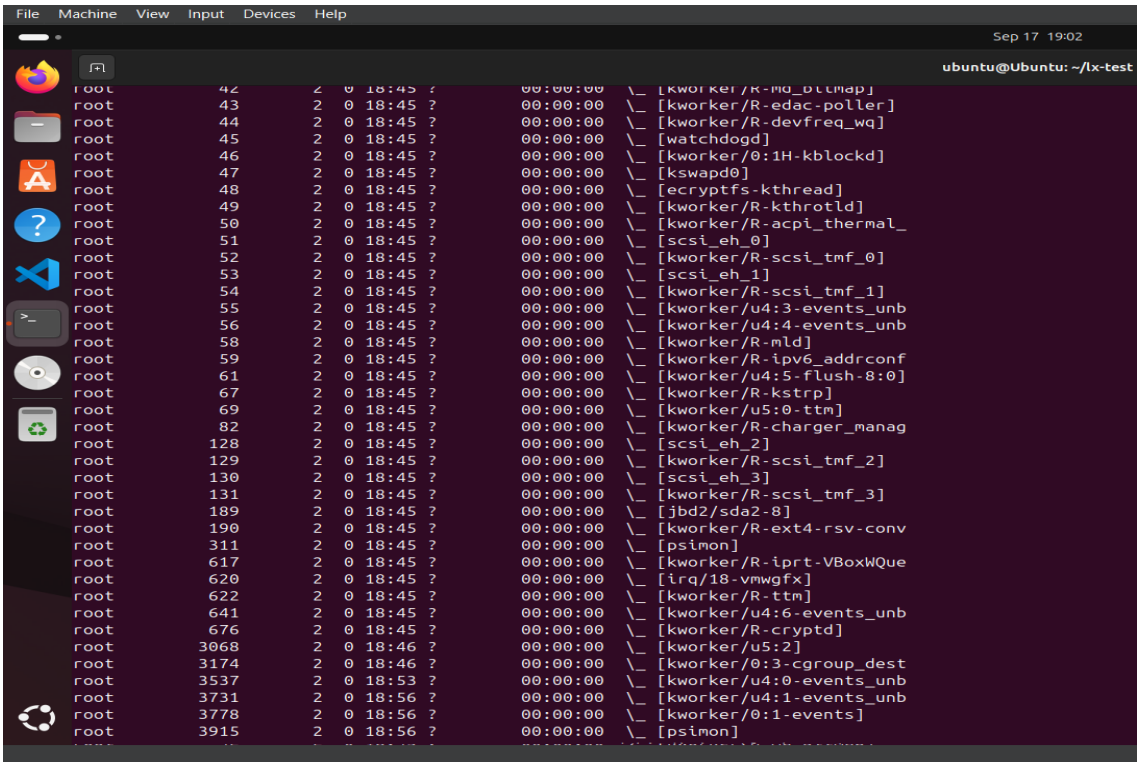

Figure 3.9: Part F #24 Tree View 2

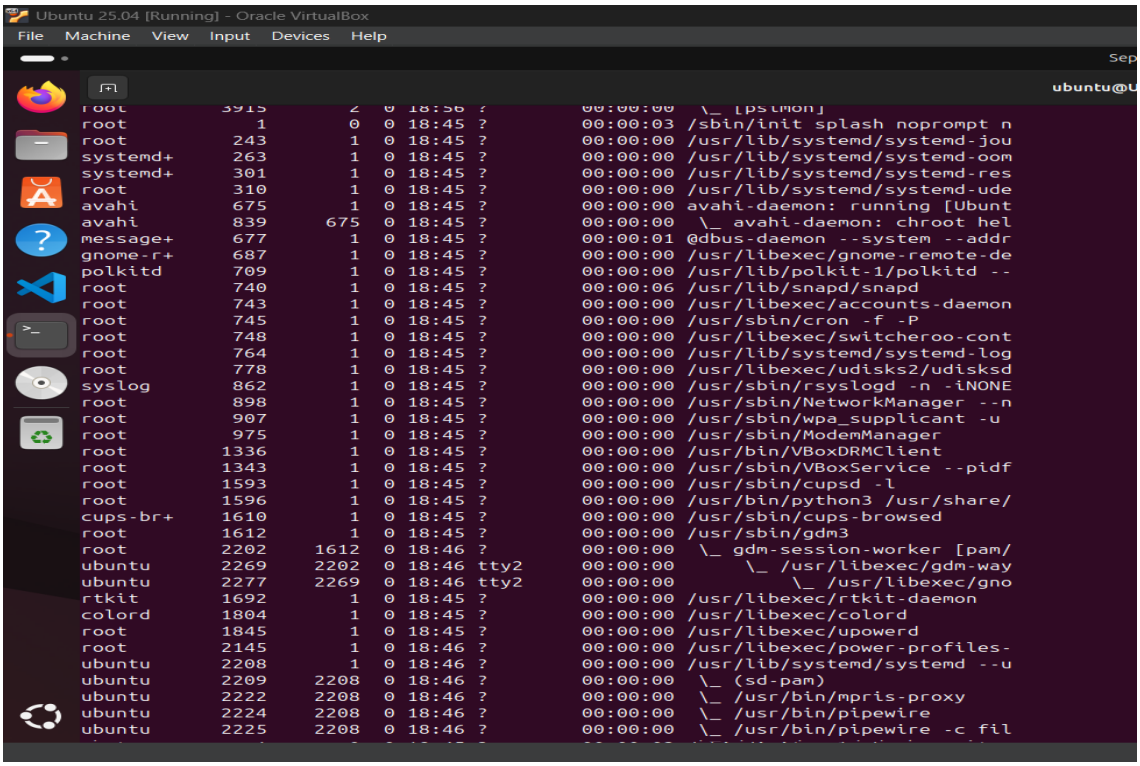

Figure 3.10: Part F #24 Tree View 3

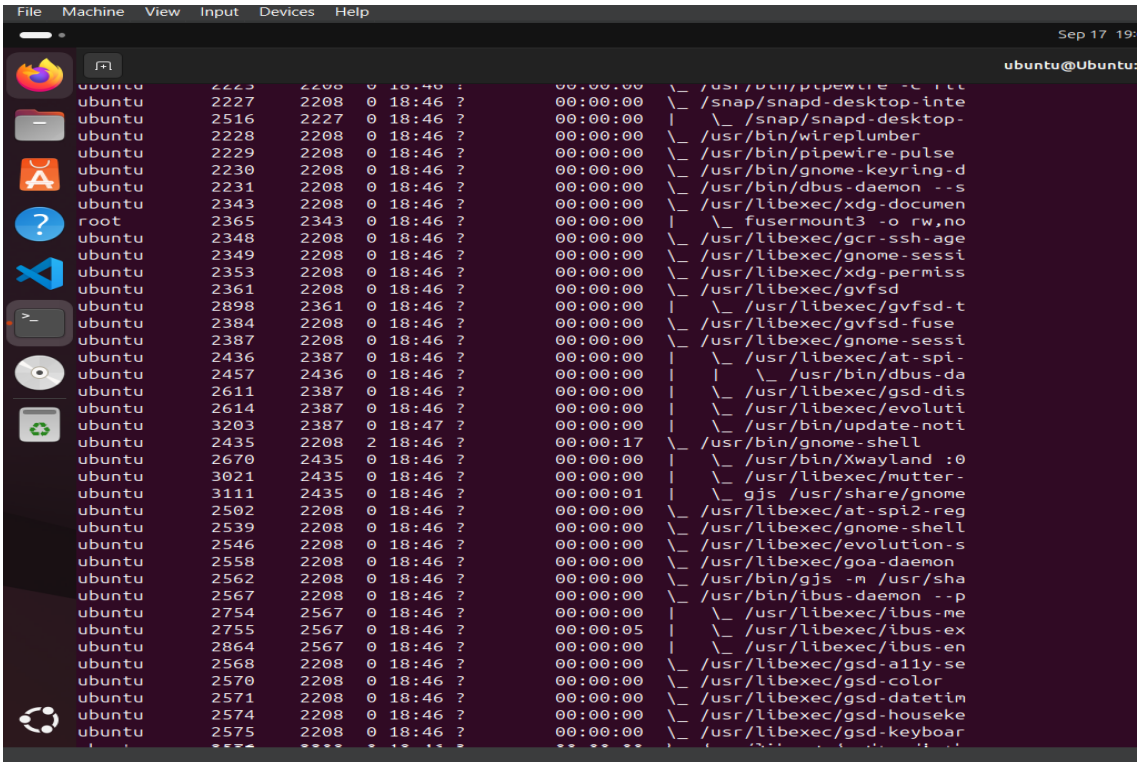

Figure 3.11: Part F #24 Tree View 4

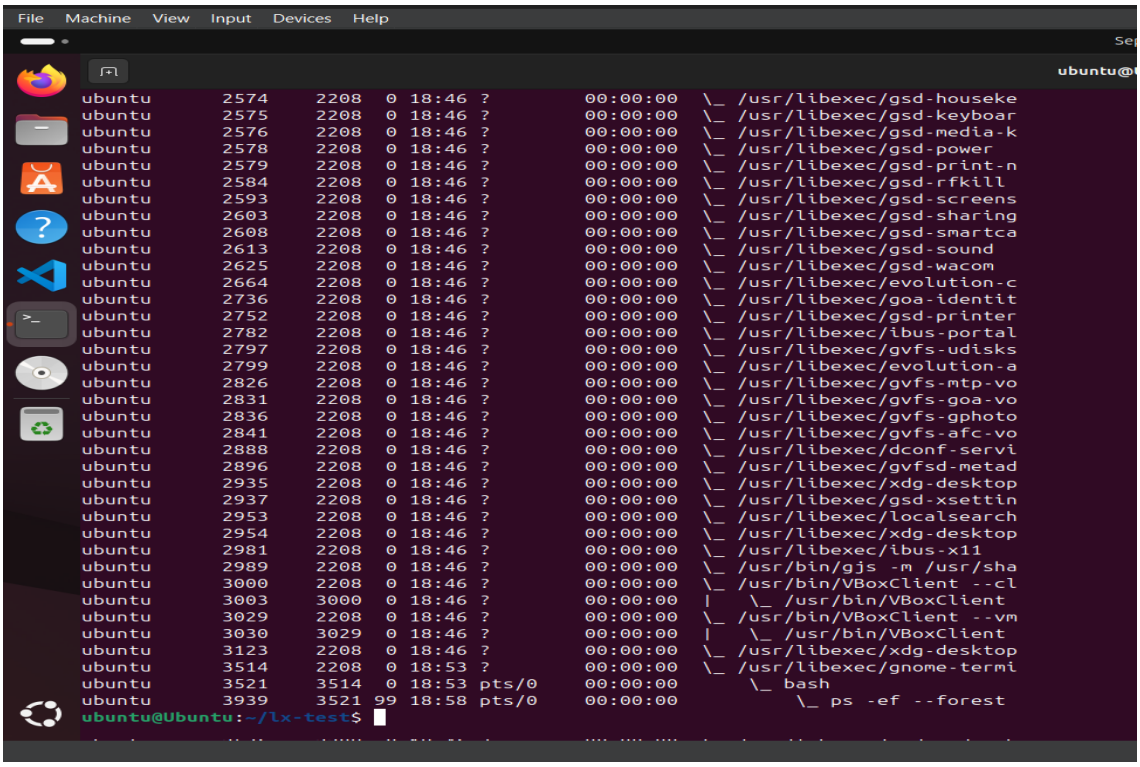

Figure 3.12: Part F #24 Tree View 5

- 25:

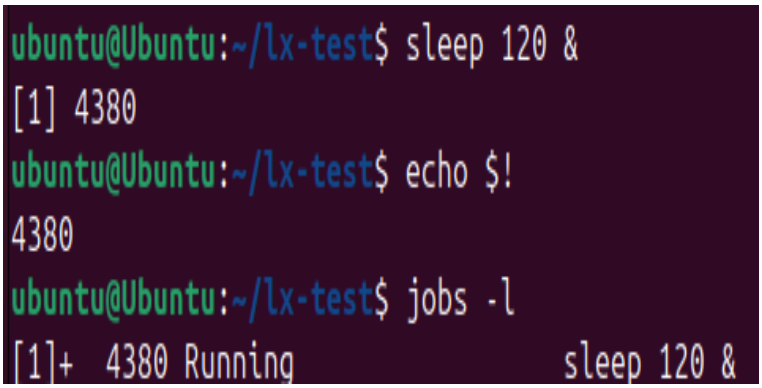

Figure 3.13: Part F #25 Sleep 120 in the background and its PID

- 26:

```
ubuntu@Ubuntu:~/lx-test$ pkill -TERM -u "$USER" sleep
ubuntu@Ubuntu:~/lx-test$ pgrep -a sleep || echo "no sleep processes found"
no sleep processes found
```

Figure 3.14: Part F #26 TERM signal

- 27:

```
ubuntu@Ubuntu:~/lx-test$ ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head -n 6
    PID    PPID CMD                         %MEM %CPU
   2435    2208 /usr/bin/gnome-shell        11.1  3.9
   3021    2435 /usr/libexec/mutter-x11-fra  3.2  0.0
   3003    3000 /usr/bin/VBoxClient --clipb  3.0  0.0
   2937    2208 /usr/libexec/gsd-xsettings   2.4  0.0
   3111    2435 gjs /usr/share/gnome-shell/  2.0  0.1
```

Figure 3.15: Part F #27 Top 5 Processes

3.7 Part G: Archiving & Compression

- 28:

```
tar czf src.tar.gz
```

- 29:

```
tar -tf src.tar.gz
```

- 30:

```
tar -xzf src.tar.gz -C tmp src/file2.txt
```

3.8 Part H: Networking & System Info

- 31:

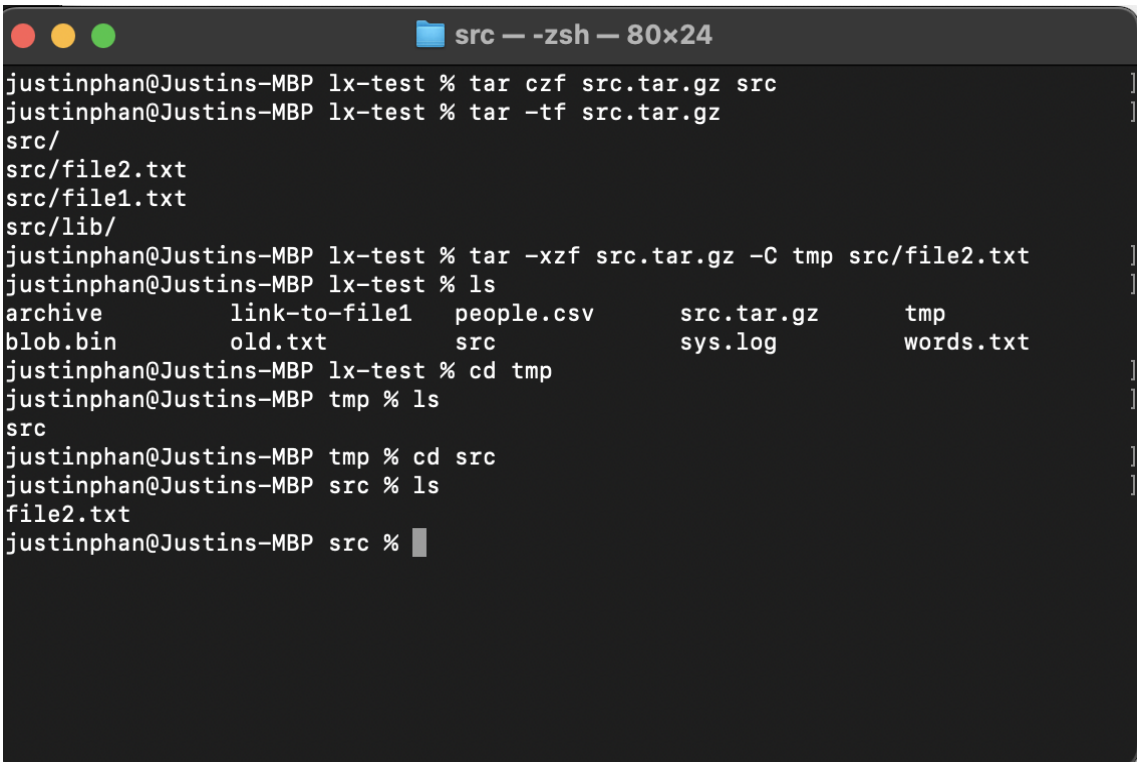

Figure 3.16: Part G Terminal Output

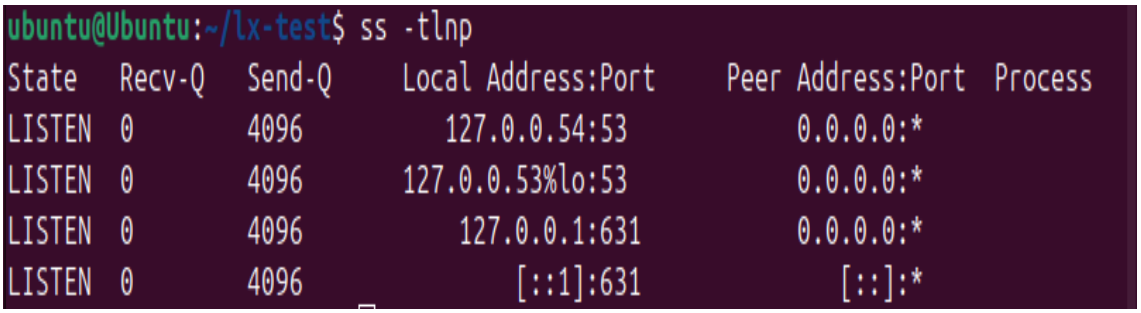

Figure 3.17: Part H #31 TCP sockets with associated PIDs

- 32:

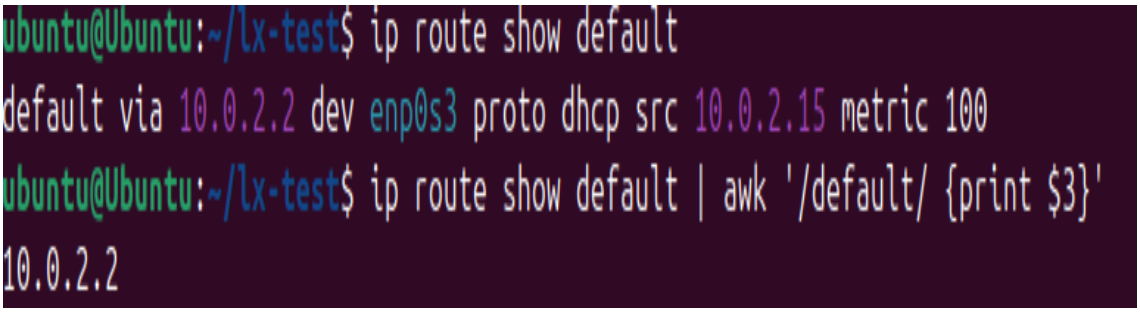

Figure 3.18: Part H #32 Default Route (gateway) in a concise form

- 33:

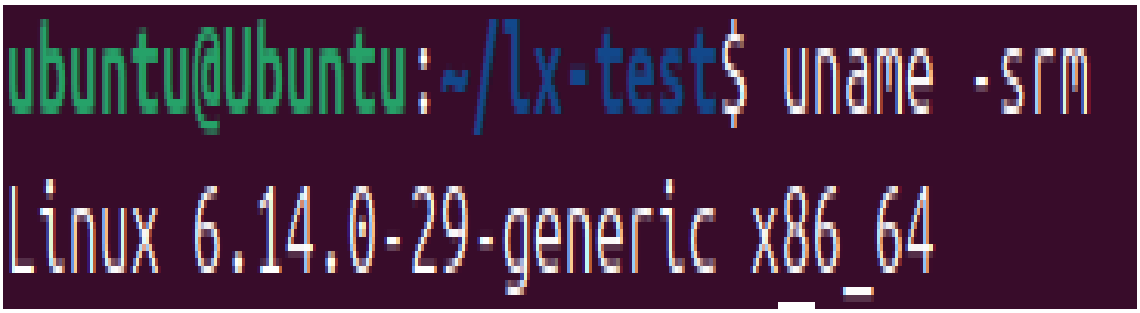

Figure 3.19: Part H #33 Kernel name, Release, and Machine Architecture

- 34:

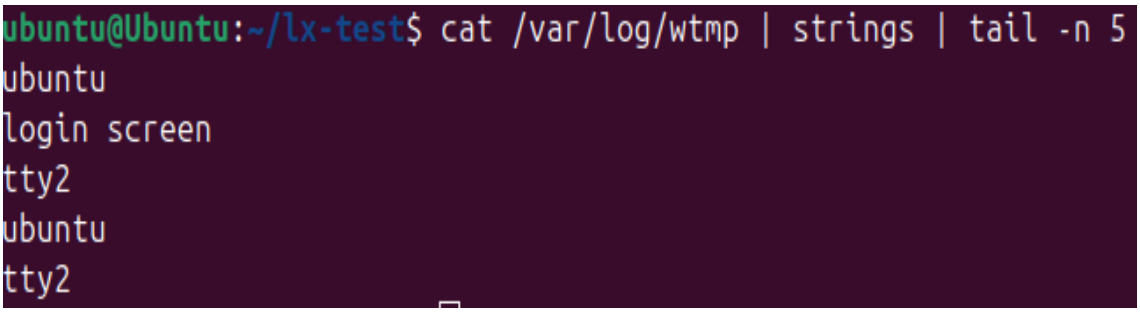

Figure 3.20: Part H #34 Last 5 successful logins on the system

3.9 Part I: Package & Services (Debian/Ubuntu)

- 35:

```
brew list --versions coreutils

```

- 36:

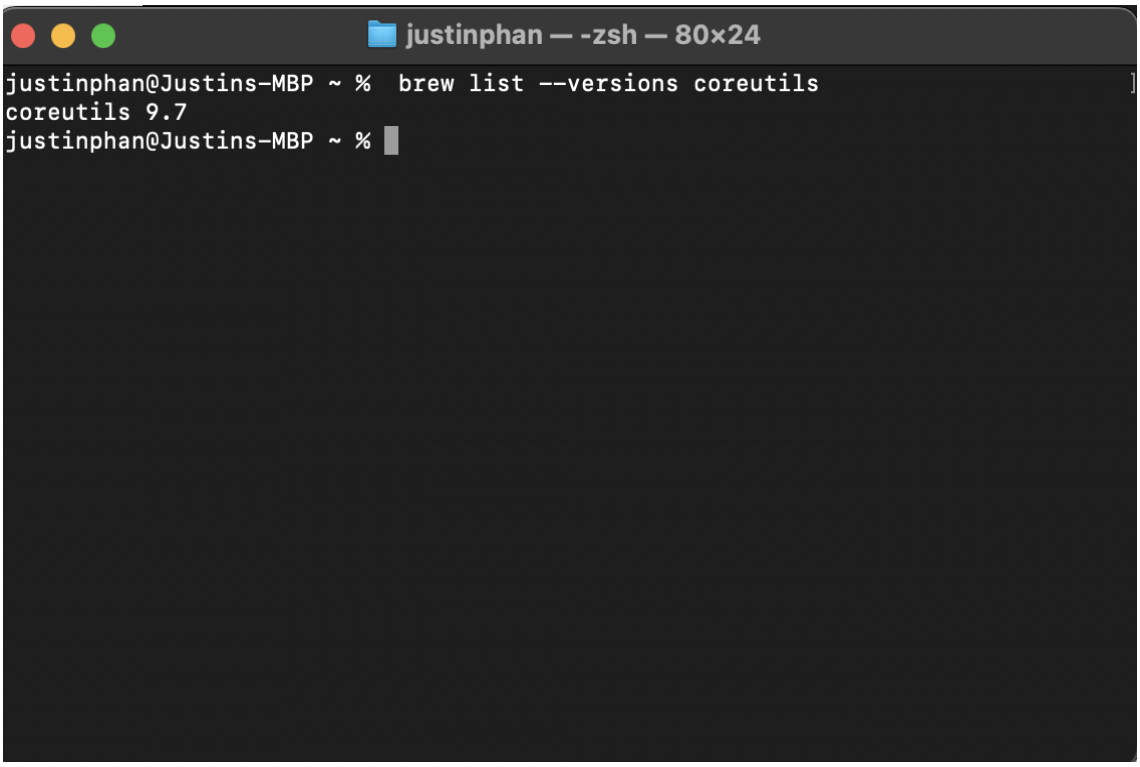

Figure 3.21: Part I #35 Terminal Output showing the installed version of package coreutils

```
brew search ripgrep
```

- 37:

```
systemctl status cron | grep "Active."
```

3.10 Part J: Bash & Scripting

- 38:

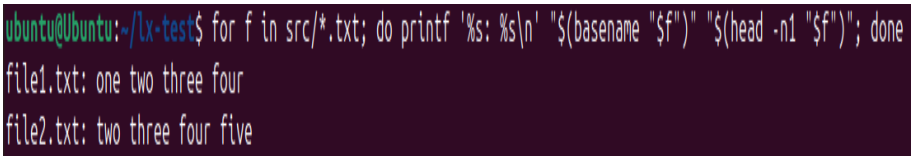

Figure 3.24: Part J #38 One-liner that loops over

- 39:

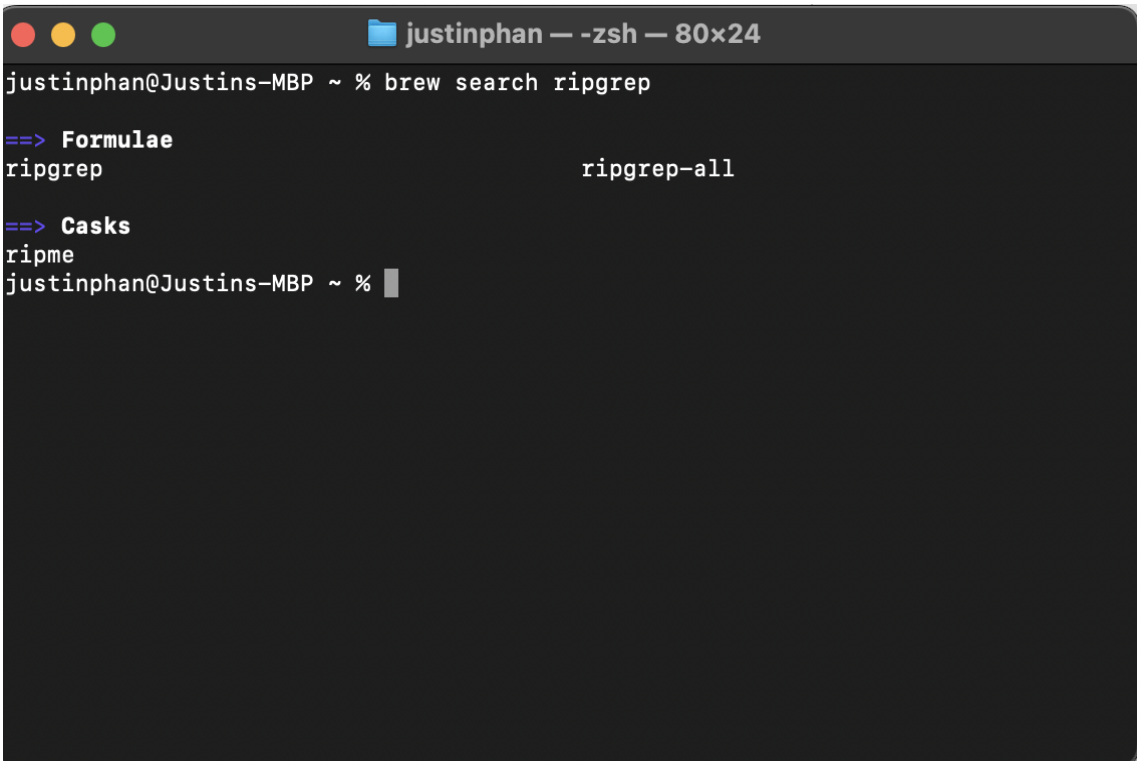

Figure 3.22: Part I #36 Terminal Output showing all available packages whose names contain ripgrep

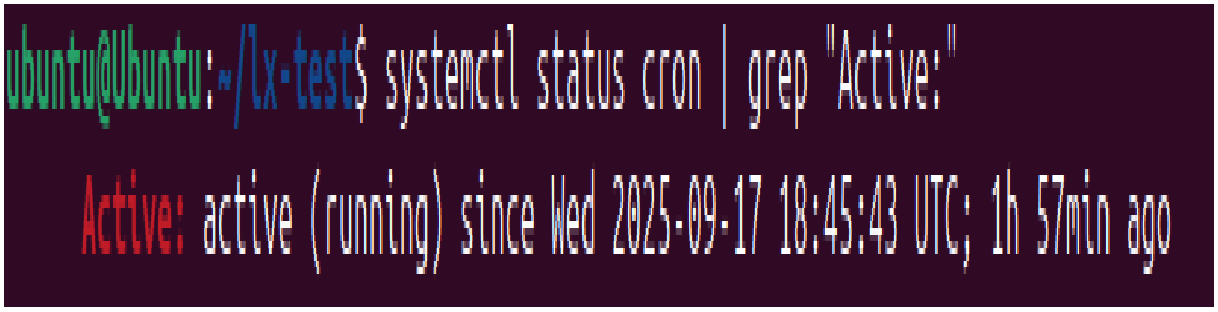

Figure 3.23: Part I #37 Terminal Output

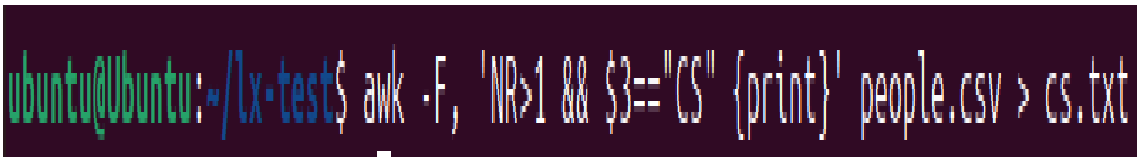

Figure 3.25: Part J #39 A command that exports CSV rows

- 40:

```
ubuntu@Ubuntu:~/lx-test$ export X=42
ubuntu@Ubuntu:~/lx-test$ echo "$X"
42
ubuntu@Ubuntu:~/lx-test$ unset X
ubuntu@Ubuntu:~/lx-test$ env | grep -w '^X' || echo "X is unset"
X is unset
ubuntu@Ubuntu:~/lx-test$
```

Figure 3.26: Part J #40 Create a variable X with value 42

Chapter 4

Project Proposal

– Charles, Justin, Benedict, Jacky

Project Title: Dicey DevOps: A Luck-Based Probability Game

Project Description: Our project is a web-based luck game that also teaches concepts of probability and the idea of risk versus reward. Players will roll dice and place bets on different outcomes such as totals, pairs, triples, or exact numbers. They can choose to re-roll or lock dice, which adds strategic choices between safer but lower-value options and riskier plays with higher potential rewards. The game will also feature occasional "event rounds" where special conditions apply (such as bonus payouts or inverted win conditions). After each round, the game will provide insights into the actual probability of success and expected value, helping players better understand chance, statistics, and decision-making.

Sample tasks include designing the dice roll system with fair randomness, building a leader board to track high scores, and implementing "learning mode" features that explain probabilities to the player. Some tasks we have brainstormed include implementing the dice roll system, building a leader board, and creating a "learning mode" that explains probability insights to players. We will also design a dashboard to track both system health and game-related metrics such as win/loss ratios, re-roll usage, and outcome distributions.

For the DevOps side, we plan to use version control, testing, deployment pipelines, and monitoring tools. Examples may include GitHub for source control, Gitlab CI/CD pipelines to automate builds and deployments, PostgreSQL or another database for storing results, containerization with Docker, infrastructure as code tools for environment setup, and monitoring solutions such as Prometheus and Grafana. We are still discussing the exact tech stack as a team, but our goal is to keep the setup lightweight and easy to extend while still allowing us to compare multiple DevOps tools as required.

Chapter 5

AWS Deployment

– Charles, Justin, Benedict, Jacky

Live URL: `https://zjnbvfjpug.us-east-1.awsapprunner.com`

Overview

We deployed the *Two Buttons* website as a containerized service on AWS. The pipeline is:

1. Authenticate the AWS CLI via **SSO** (IAM Identity Center).
2. Build a Docker image locally and push it to **Amazon ECR**.
3. Deploy from ECR to a managed container runtime using **AWS App Runner**.

Why App Runner? For a small stateless web app, App Runner removes the need to manage ECS tasks/services, load balancers, or EC2 capacity. It auto-builds or pulls from ECR, provisions HTTPS and scaling, and exposes a public URL with minimal ops overhead (good for a course project).

Project root used for the build `C:\Users\benma\Documents\docker-examples-benedict\docker-examples\color-buttons-app`

Prerequisites

- Windows 10/11 with **Docker Desktop** and **AWS CLI v2**.
- **SSO** set up in the AWS Console (IAM Identity Center) with an AdminAccess permission set (temporary for the lab).
- Overleaf configured to compile with `minted` (shell escape enabled).

Authenticate the AWS CLI (SSO)

```
aws configure sso
aws sso login --profile default
aws sts get-caller-identity --profile default
# Confirm the returned Account ID is yours and an SSO role is assumed.
```

Build, Tag, and Push the Image to Amazon ECR

```
# Go to the project folder
cd "C:\Users\benma\Documents\docker-examples-benedict\docker-examples\color-butt
 ↪  ons-app"

# Environment
$Env:AWS_REGION="us-east-1"
$Env:ECR_REPO="color-buttons-app"
$Env:IMAGE_TAG="v1"
$Env:AWS_ACCOUNT_ID=(aws sts get-caller-identity --query Account --output text
 ↪  --profile default)

# Create the ECR repo if missing
aws ecr describe-repositories --repository-names $Env:ECR_REPO --region
 ↪  $Env:AWS_REGION --profile default *> $null ; if ($LASTEXITCODE -ne 0) {
  aws ecr create-repository --repository-name $Env:ECR_REPO `
    --image-scanning-configuration scanOnPush=true `
    --region $Env:AWS_REGION --profile default
}

# Log in Docker to ECR
aws ecr get-login-password --region $Env:AWS_REGION --profile default |
  docker login --username AWS --password-stdin
   ↪  "$($Env:AWS_ACCOUNT_ID).dkr.ecr.$($Env:AWS_REGION).amazonaws.com"

# Build, tag, and push (force linux/amd64 for App Runner build fleet
 ↪  compatibility)
docker build --platform linux/amd64 -t "$($Env:ECR_REPO):$($Env:IMAGE_TAG)" .
docker tag  "$($Env:ECR_REPO):$($Env:IMAGE_TAG)" `
            "$($Env:AWS_ACCOUNT_ID).dkr.ecr.$($Env:AWS_REGION).amazonaws.com/$($
             ↪  Env:ECR_REPO):$($Env:IMAGE_TAG)"
docker push "$($Env:AWS_ACCOUNT_ID).dkr.ecr.$($Env:AWS_REGION).amazonaws.com/$($
 ↪  Env:ECR_REPO):$($Env:IMAGE_TAG)"
```

Deploy with AWS App Runner (Console)

1. **Create service** → **Source**: *Container registry* → *Amazon ECR*.
   Choose repository `color-buttons-app` and tag `v1`.

2. **Service name**: `color-buttons-app` **Port**: `3000`.
3. **ECR access role**: *Create new service role* (let App Runner pull from ECR).
4. **Health check**: `HTTP` on path `/` (timeout 5s, interval 10s).
5. Click **Create & Deploy**; wait for *Status: Running* and note the *Default domain*.

Verification

```
# Expect HTTP/2 200 (or similar)
curl -I https://zjnbvfjpug.us-east-1.awsapprunner.com
```

Manually verify the page renders and both buttons switch background color (blue/red).

Operating the Service (Logs, Scaling, Rollback)

- **Logs:** In App Runner → *Logs* tab to view system/app logs.
- **Scaling:** Default concurrency is 100 requests/instance, min 1, max 25 instances.
- **Re-deploy:** Push the same tag and choose *Actions* → *Deploy* for manual trigger services.
- **Rollback:** Keep prior image tags; re-point the service to the last known-good tag and redeploy.

Cost Guardrails & Cleanup

App Runner and ECR are pay-as-you-go. To avoid charges after grading:

1. In **App Runner**: *Actions* → *Pause* or *Delete* the service.
2. In **ECR**: delete the image(s) and (optionally) the repository.

(We also set an AWS Budget alarm earlier to notify on any unexpected spend.)

Figure

Class-based JavaScript Refactor

We replaced the old function-based handlers with an class that encapsulates all behavior (buttons, events, and background updates). Only the public assets changed (`public/index.html`, `public/app.js`); the server continues to serve `public/` and listen on `0.0.0.0:3000`.

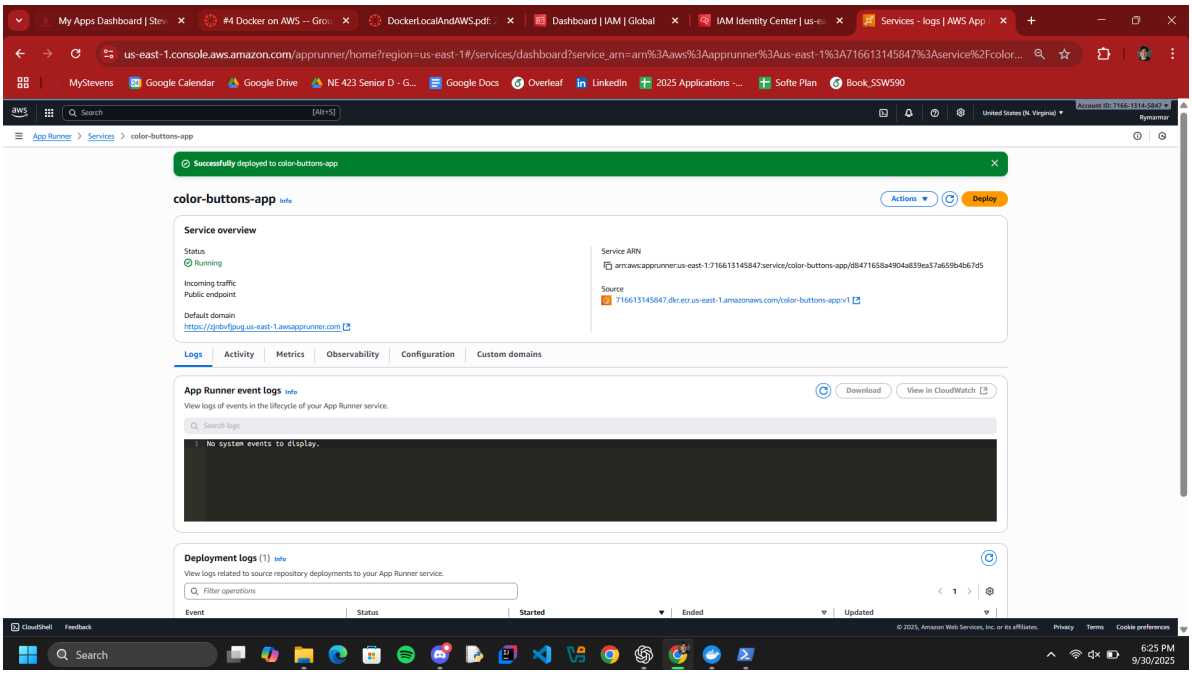

Figure 5.1: App Runner service in *Running* state with the default domain.

Updated `public/index.html`

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Two Buttons</title>
</head>
<body>
  <h1>Two Buttons</h1>
  <button id="blueBtn">Blue</button>
  <button id="redBtn">Red</button>

  <script src="app.js"></script>
</body>
</html>
```

New `public/app.js` (class-based)

```
class ColorButtonsApp {
  constructor() {
    this.$blue = document.getElementById("blueBtn");
    this.$red  = document.getElementById("redBtn");
    this.bindEvents();
  }
```

```
  bindEvents() {
    this.$blue.addEventListener("click", () => this.setBg("steelblue"));
    this.$red .addEventListener("click", () => this.setBg("crimson"));
  }
  setBg(color) { document.body.style.backgroundColor = color; }
}
window.addEventListener("DOMContentLoaded", () => new ColorButtonsApp());
```

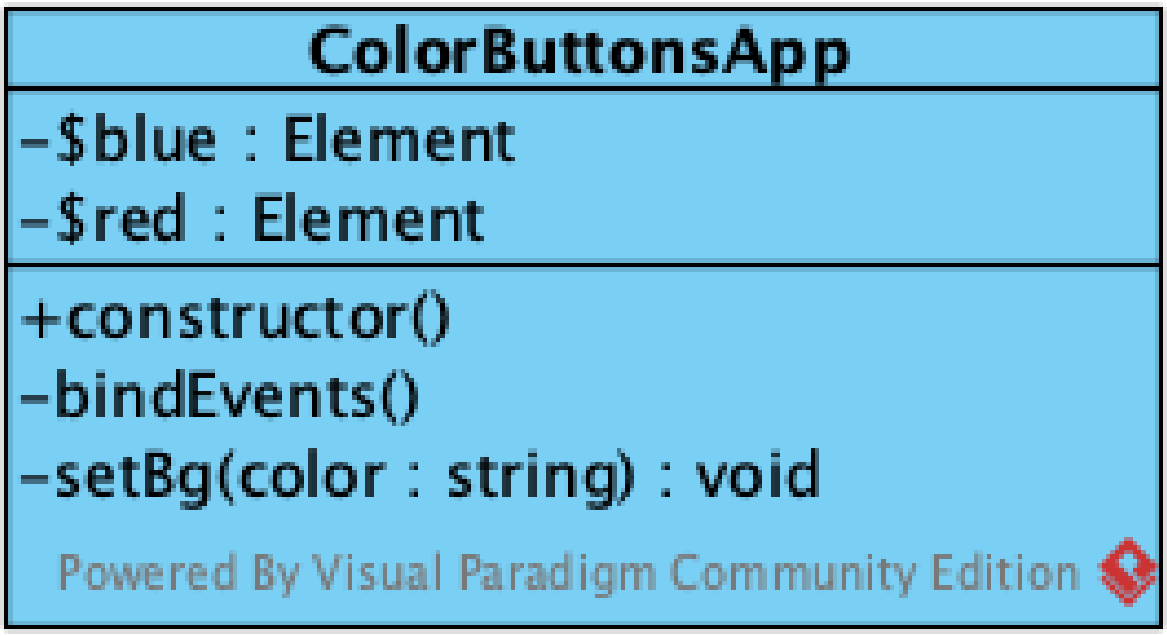

Figure 5.2: Class Diagram for App.js

Server.js

```
import express from "express";
import path from "path";
import { fileURLToPath } from "url";

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const app = express();
const PORT = process.env.PORT || 3000;

// Serve static files from public/
app.use(express.static(path.join(__dirname, "public")));

app.listen(PORT, "0.0.0.0", () => {
```

```
  console.log(`Server running at http://0.0.0.0:${PORT}!`);
});
```

package.json

```
{
  "name": "color-buttons-app",
  "version": "1.0.0",
  "type": "module",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

Result. Clicking **Blue** or **Red** now triggers methods on a single `ColorButtonsApp` instance, keeping the global scope clean and making the behavior easy to unit test or extend.

Chapter 6

LaTeX Docker

– Charles, Justin, Benedict, Jacky

6.0.1 Project Directory Setup

- Create a folder docker-latex
- Start docker
- Make sure docker is running by using docker run hello-world
- cd into that folder directory

6.0.2 Docker Commands

- Create a Dockerfile with the content below

```
FROM debian:bullseye-slim

ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update && \
    apt-get install -y \
        texlive-latex-base \
        texlive-latex-recommended \
        texlive-fonts-recommended \
        texlive-latex-extra \
        make \
        && apt-get clean && rm -rf /var/lib/apt/lists/*

WORKDIR /doc

CMD ["pdflatex", "main.tex"]
```

- Run nano main.tex and paste your desired LaTeX content
- Build the docker image

- Run the docker command
- Check your folder to see if the main.pdf file is created

```
nano main.tex
docker build -t docker-latex .
docker run --rm -v "$PWD":/doc docker-latex
```

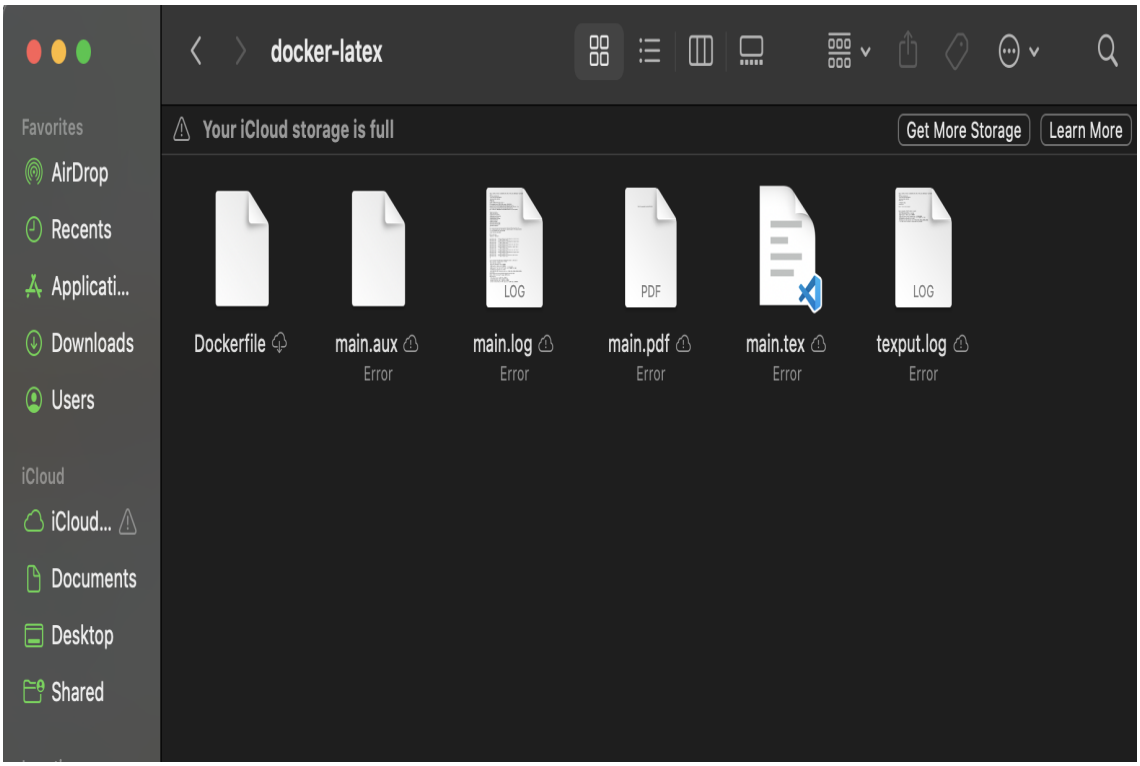

Figure 6.1: Folder containing files created from successful Docker build

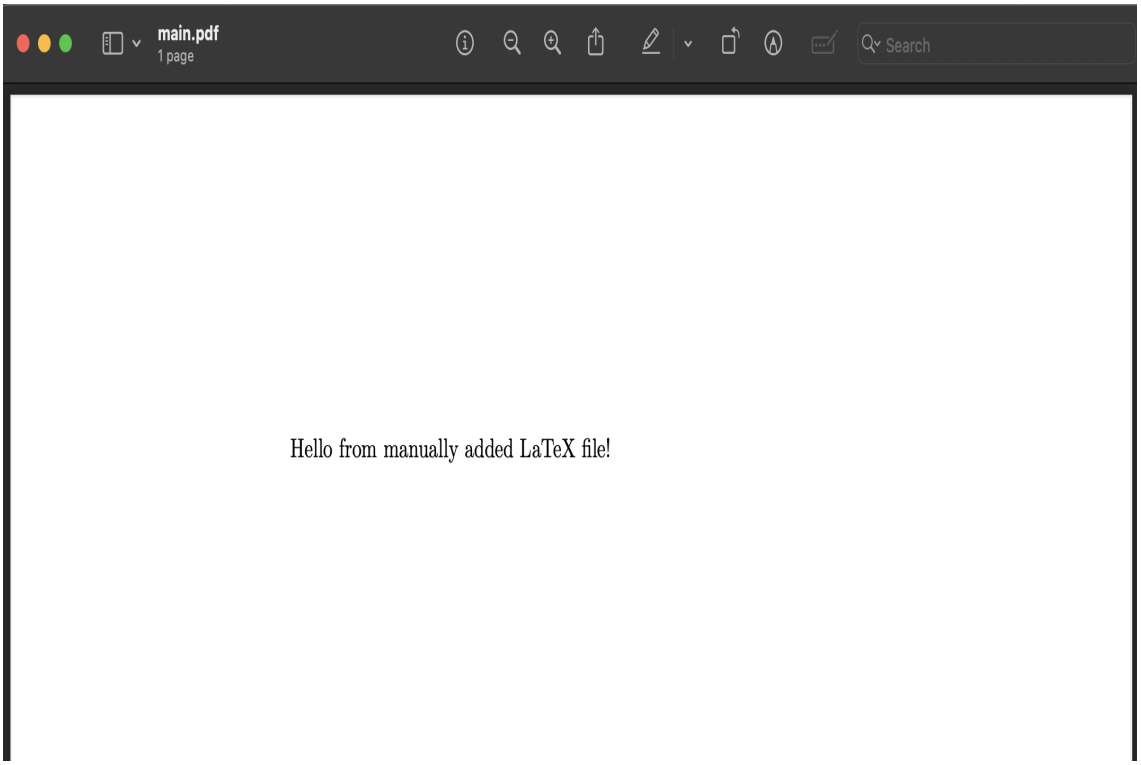

Figure 6.2: Docker compiled LaTeX document

Chapter 7

Bugzilla

– Charles, Justin, Benedict, Jacky

1. Setup Cloud Infrastructure and Access
    - Created a Digital Ocean account and created a Ubuntu Droplet VM.
    - Secured access by generating and adding an SSH key to the Droplet.
2. Prepare Container Environment
    - Cloned the Bugzilla source code.
    - Installed and fixed the missing dependencies (Docker Compose and the Docker service daemon) needed to run containers.
3. Deploy Application Containers
    - Used Docker Compose to launch two linked services: the Bugzilla web application container and the MariaDB database container.
    - Ensured the application's internal network port was mapped to the Droplet's external port 8080.
4. Finalize Configuration
    - Executed the required Bugzilla setup script (checksetup.pl) inside the running web container to build the database schema and verify system readiness.
5. Access and Admin Creation
    - Confirmed the application was accessible in a web browser at the public IP and port (http://174.138.69.132.8080).
    - Completed the final step by creating the administrator account via the web interface.
    - Deleted the Droplet (which is why the link might not work anymore) to avoid any unnecessary billing.

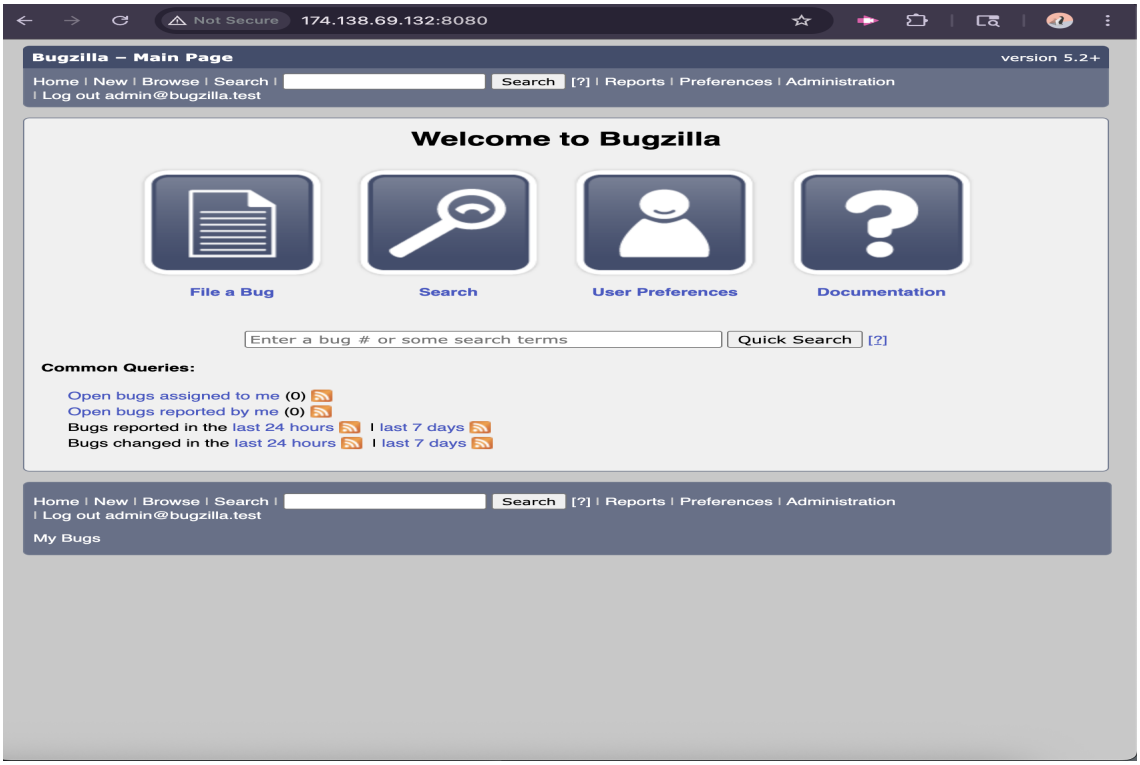

Figure 7.1: Bugzilla Container Running Screenshot

Chapter 8

Overleaf

– Charles, Justin, Benedict, Jacky

1. Setup Cloud Infrastructure and Access
    - Created a Digital Ocean Droplet (VM) running Ubuntu.
    - Secured access to the server by using the SSH protocol through the local terminal, which was made in the Bugzilla step.
    - Resolved an initial SSH access issue to gain root privileges on the Droplet.
2. Prepare Container Environment
    - Installed the necessary container tools (Docker Engine and Docker Compose V1), resolving dependency issues with the correct package name.
    - Cloned the Overleaf Toolkit source code into the overleaf-ce directory.
    - Edited the config/overleaf.tc file to set the application's public-facing address and listen on the appropriate IP/Port:
        - Set OVERLEAF_LISTEN_IP=0.0.0.0 and OVERLEAF_PORT=80
3. Deploy Application Containers
    - Launched the linked services (sharelatex, mongo, and redis) using the Toolkit's wrapper script "bin/up -d".
    - Configured the host firewall (UFW) to allow external HTTP traffic on Port 80, exposing the application to the public internet.
4. Access and Admin Creation
    - Confirmed the application was accessible in a web browser at the public IP and port (http://104.236.74.225).
    - Deleted the Droplet (which is why the link might not work anymore) to avoid any unnecessary billing.

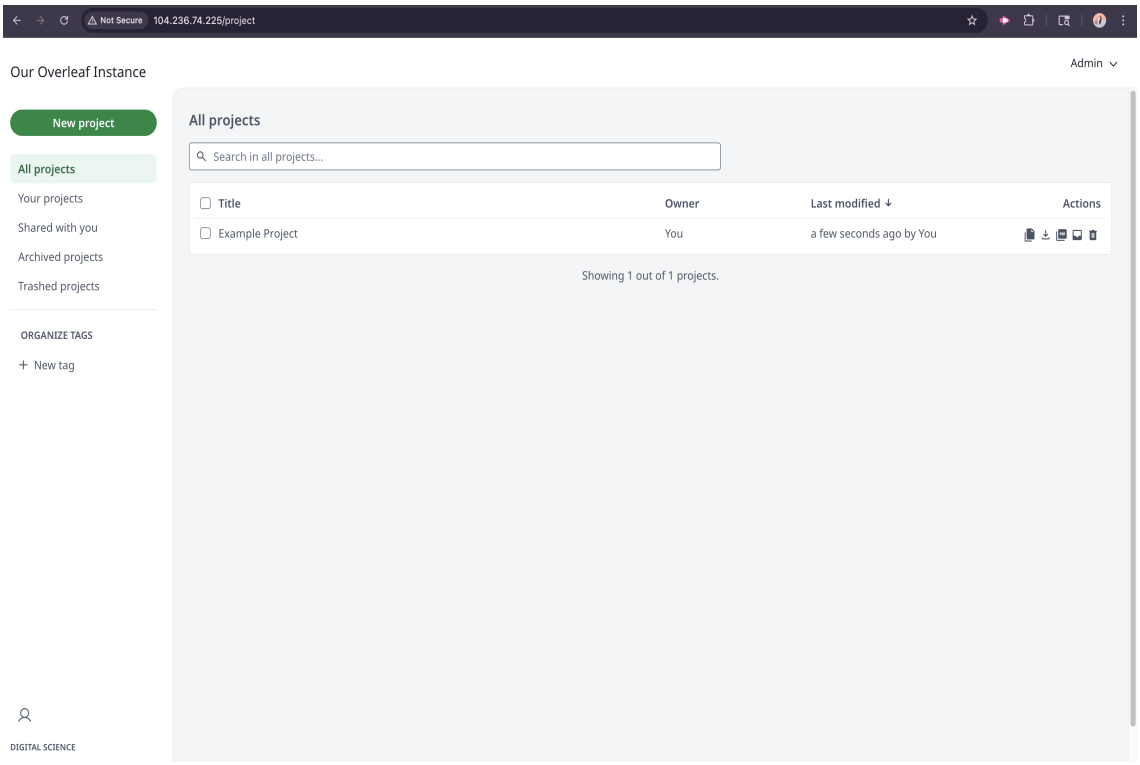

Figure 8.1: Overleaf Instance Main Menu Screenshot

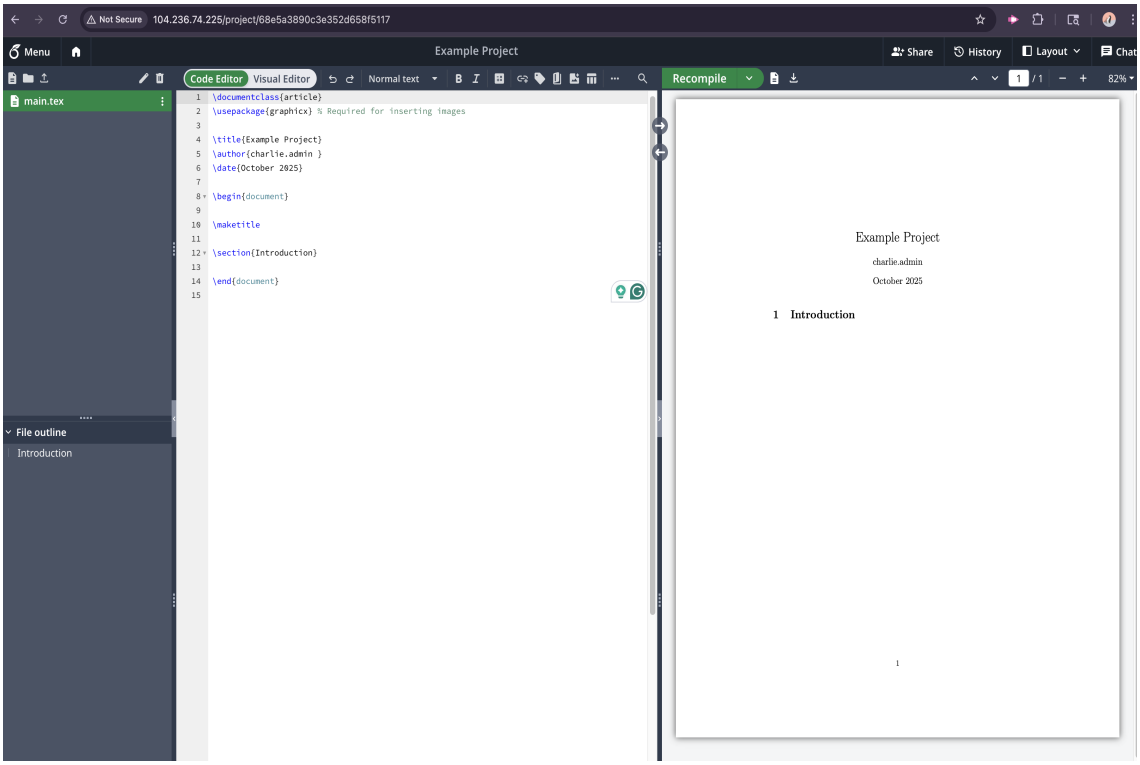

Figure 8.2: Overleaf Instance Example Project Screenshot

Chapter 9

Domain Names, SSL, and Versioning

– Charles, Justin, Benedict, Jacky

9.1 Assignment Overview

This chapter documents how our group configured a custom domain, secured it with SSL, and connected it to an Overleaf instance hosted via DigitalOcean and GitHub Pages. The work combines the DigitalOcean setup (completed by team members on macOS) with my GitHub Pages + SSL configuration. Screenshots and command examples are provided for each major step.

9.2 Step 1: Domain Registration

Our team registered the domain **rymarmar.me** using Namecheap through the GitHub Student Developer Pack (free for one year). The domain was verified, includes WHOIS privacy protection, and is configured to auto-renew.

9.3 Step 2: SSL Certificate Configuration

To secure all traffic, we enabled **HTTPS enforcement** using GitHub Pages, which automatically provisions SSL certificates through Let's Encrypt (renewed every 90 days). This provides secure HTTPS access without needing to manually install certificates.

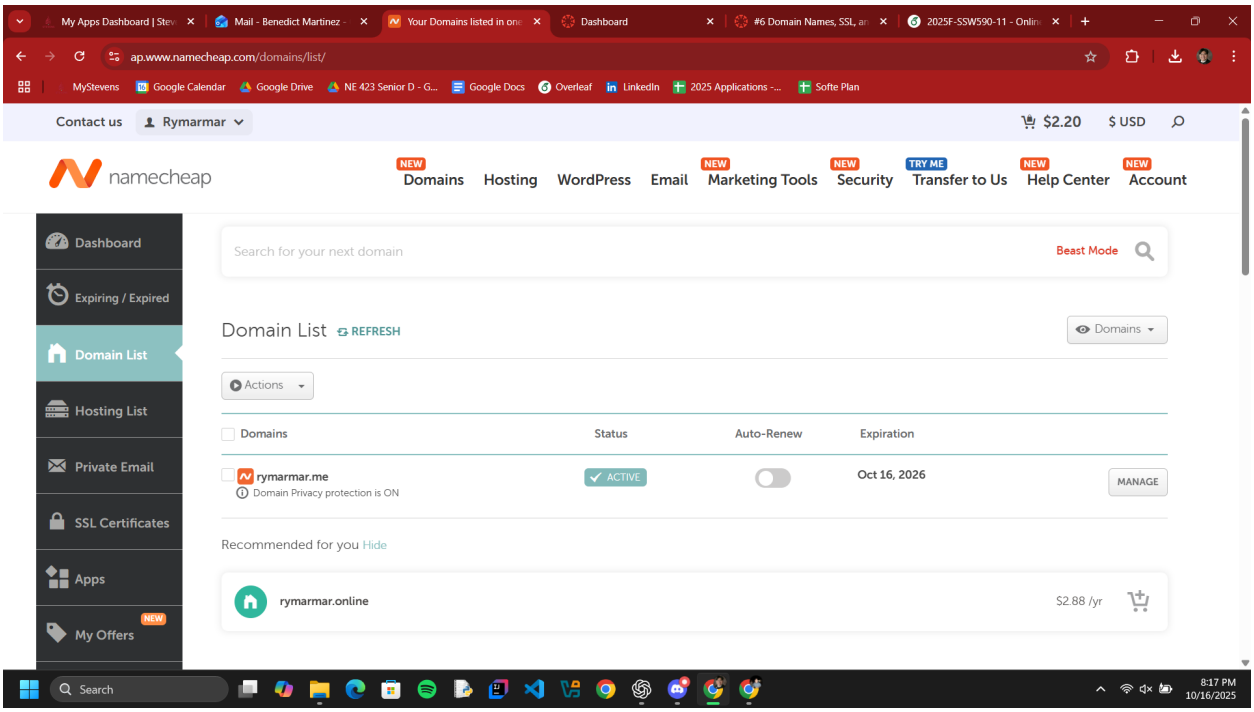

Figure 9.1: Namecheap domain list confirming registration of `rymarmar.me`.

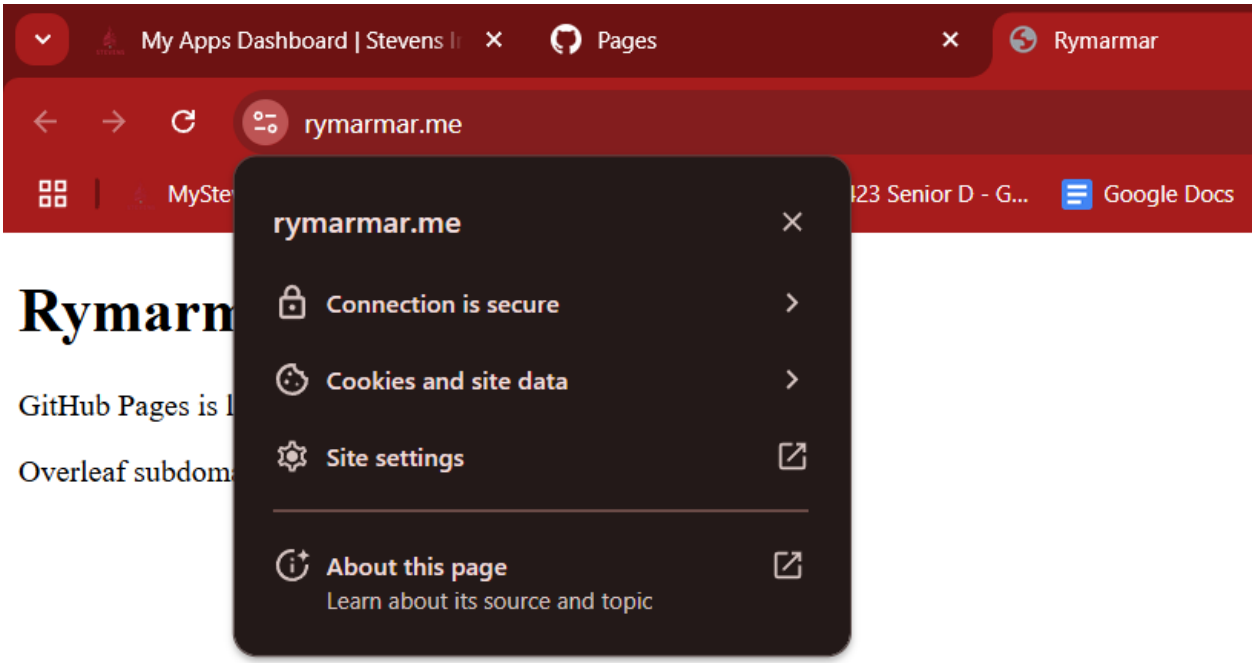

Figure 9.2: GitHub Pages settings showing HTTPS enforcement and DNS verification.

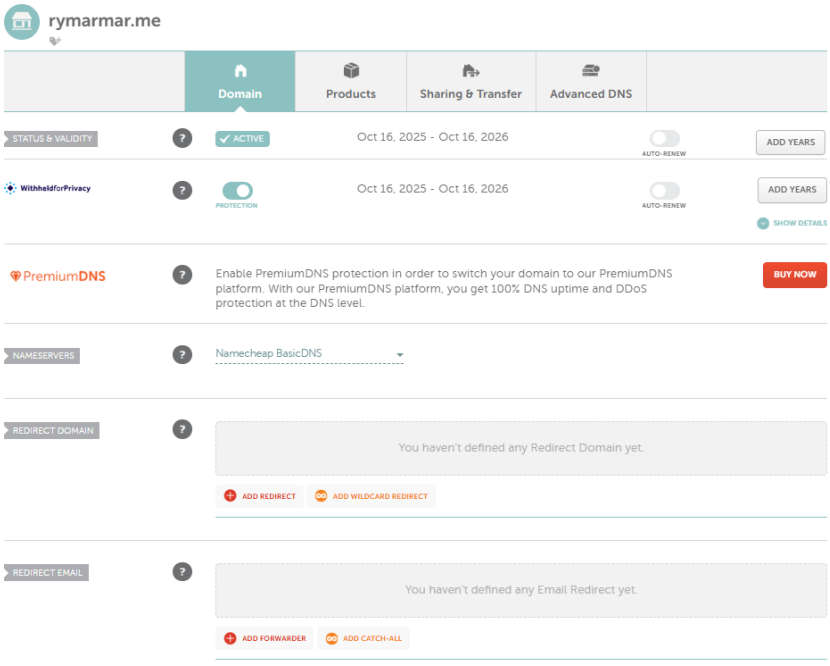

Figure 9.3: Browser confirmation that `https://rymarmar.me` is secured via SSL.

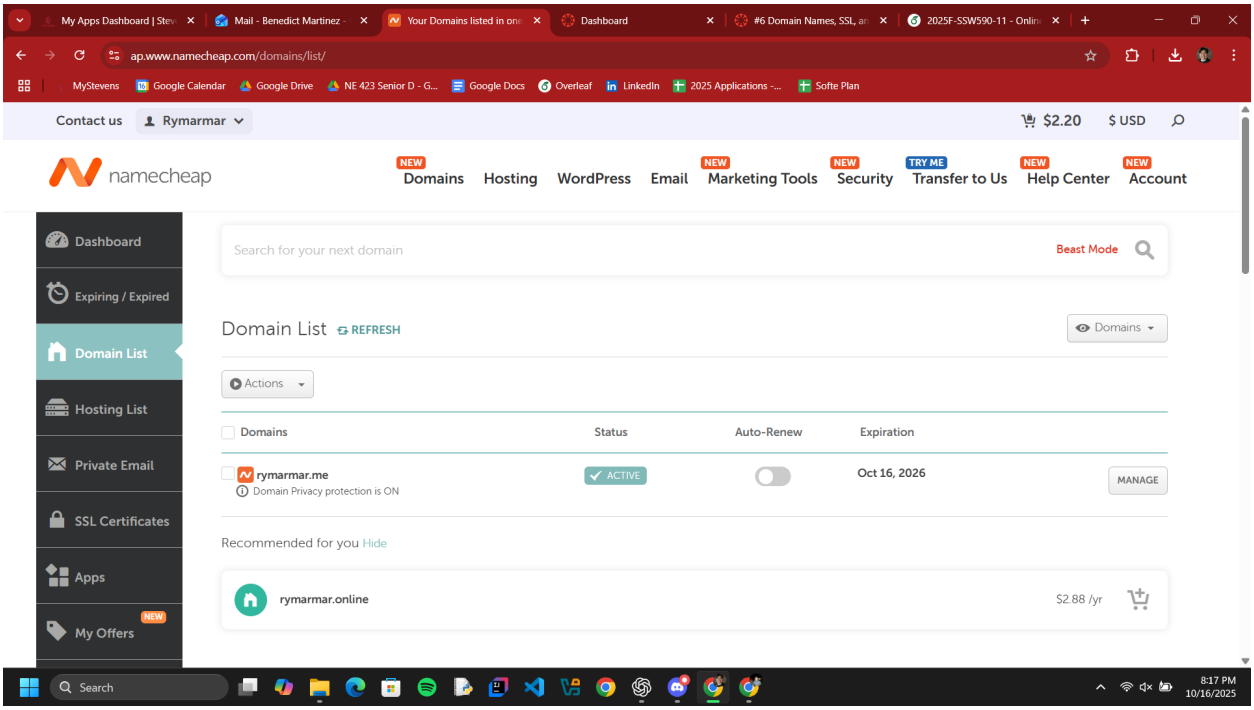

Figure 9.4: Namecheap domain list confirming registration of `rymarmar.me`.

9.4 Step 2.1: SSL Configuration using Caddy

Another option for providing SSL to the domain is using Caddy. Once installed using apt install caddy, we made a Caddyfile and modified the docker-compose file.

```
      1        services:
      2            sharelatex:
     16                volumes:
     33                    OVERLEAF_SITE_URL: "https://ssw590team11.me"
    156            caddy:
    157              image: caddy:latest
    158              container_name: caddy
    159              ports:
    160                - "80:80"
    161                - "443:443"
    162              volumes:
    163                - ./Caddyfile:/etc/caddy/Caddyfile
    164                - caddy_data:/data
    165                - caddy_config:/config
    166              depends_on:
    167                - sharelatex
    168              restart: unless-stopped
    169        volumes:
    170          mongo_data:
    171          caddy_data:
    172          caddy_config:
```

Figure 9.5: docker-compose.yml additions

```
ssw590team11.me {
        reverse_proxy sharelatex:80
}
```

Figure 9.6: Caddyfile

We also added records on Digital Ocean and Namecheap to link our droplet to the domain.

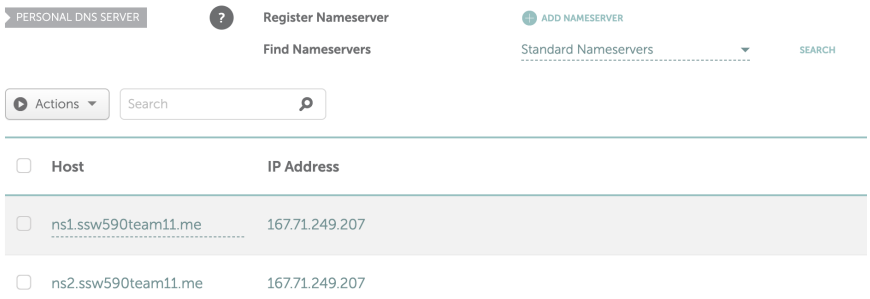

Figure 9.7: Digital Ocean DNS records linking to domain

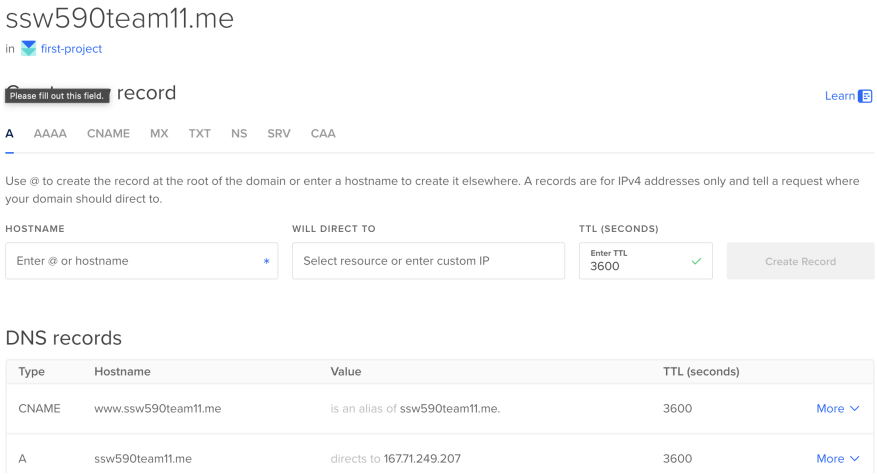

Figure 9.8: Namecheap DNS records to go to droplet IP address

9.5 Step 3: Overleaf Container Setup on DigitalOcean

Our teammates (Charles and Justin) deployed an Overleaf Community Edition container using a DigitalOcean droplet. This ensured the Overleaf instance supported all LaTeX packages and allowed testing of compilation consistency. The container was connected to our team GitHub repository and included a custom domain mapping for the subdomain.

```
# Example Docker-based deployment used on DigitalOcean
sudo docker run -d --name overleaf -p 80:80 sharelatex/sharelatex
```

This step satisfied the “configure container/image” requirement for Overleaf hosting. The configuration process was documented and replicated locally for testing.

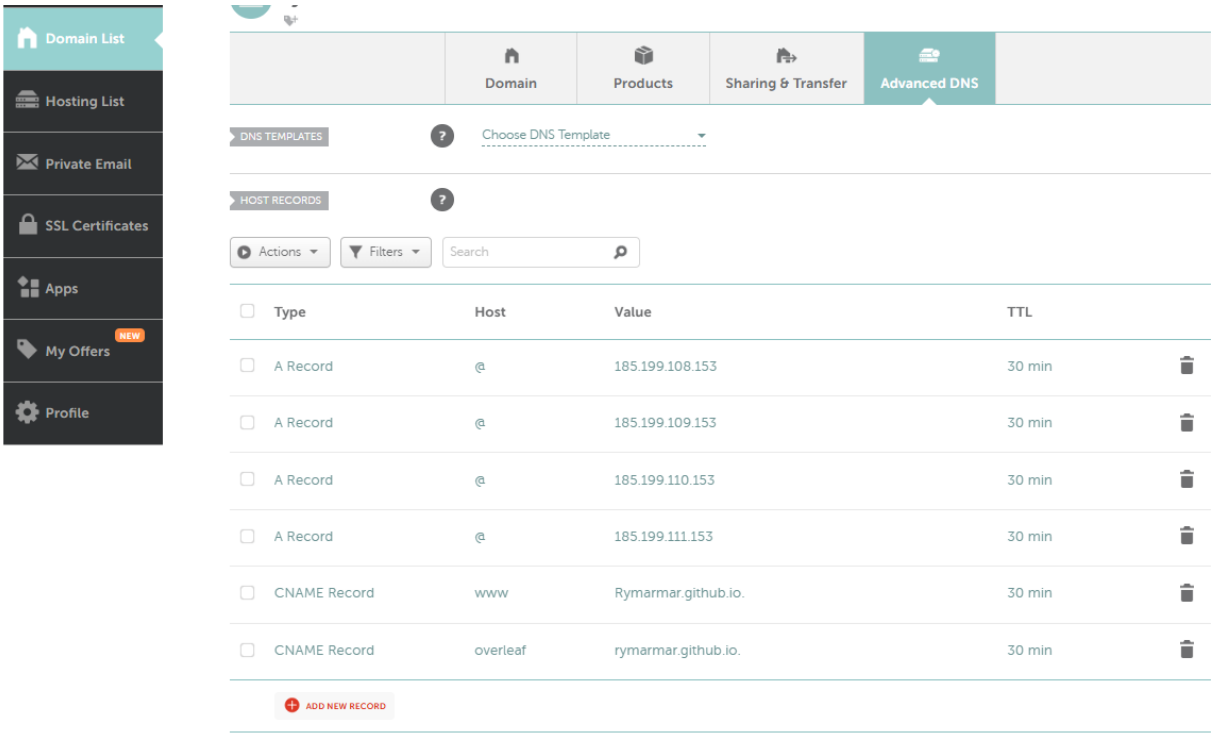

Figure 9.9: Namecheap Advanced DNS configuration showing A and CNAME records for the root and subdomains.

9.6 Step 4: GitHub Pages Integration (Benedict)

The GitHub repository `Rymarmar.github.io` was configured to serve as the main site using the custom domain `rymarmar.me`. Deployment is configured directly from the `main` branch, and HTTPS is now enforced. This serves as a simple landing page while the Overleaf container is being finalized.

The `index.html` page was published successfully. Some users experienced delays when loading the domain due to DNS propagation time, but configuration remains correct.

9.7 Step 5: Overleaf Subdomain Redirect Verification

A subdomain `overleaf.rymarmar.me` was created in Namecheap and configured to redirect to our GitHub Pages site. This verifies that DNS records for the Overleaf subdomain resolve correctly.

9.8 Step 6: Overleaf–GitHub Sync (In Progress)

Overleaf's paid Git integration feature is unavailable for free users, so synchronization is currently being replicated manually using Git commands. The workflow allows us to update Overleaf projects locally and push them to GitHub for version tracking.

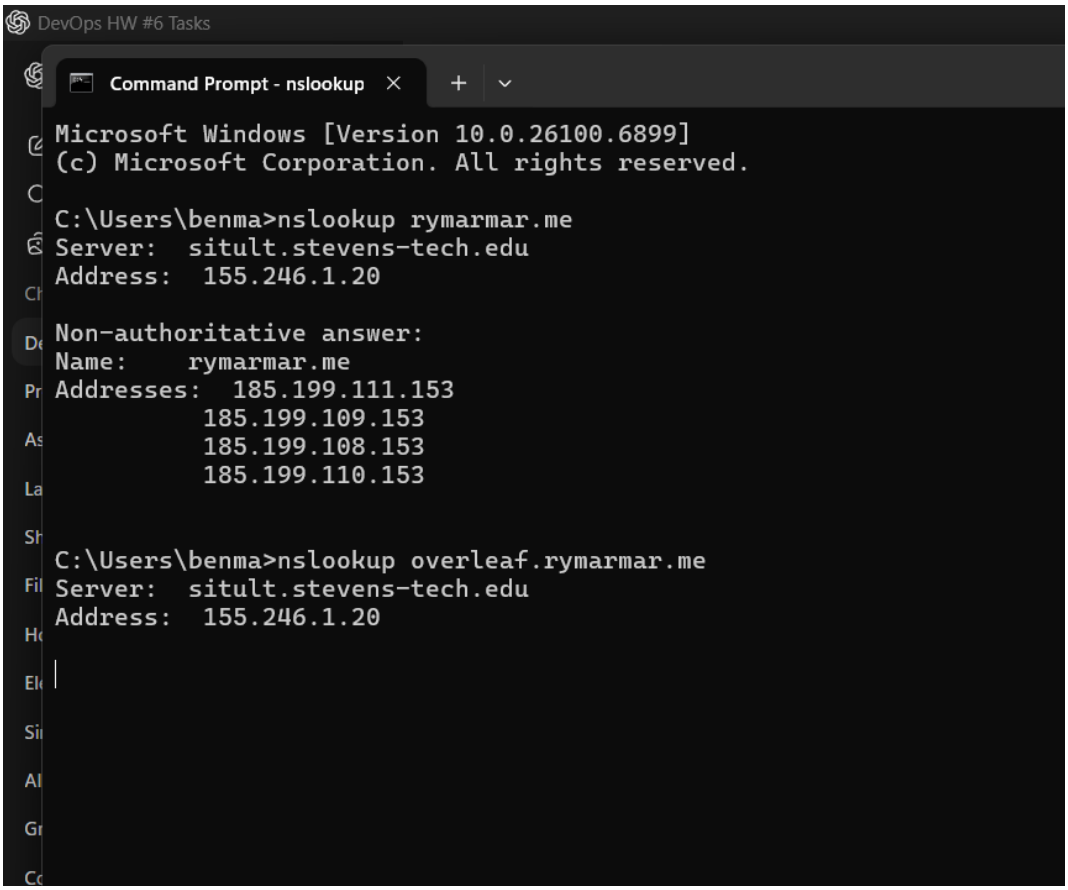

Figure 9.10: Verification using `nslookup` confirming successful DNS and subdomain resolution.

```
git clone https://github.com/Rymarmar/Overleaf.Rymarmar.me.git
cd Overleaf.Rymarmar.me
latexmk -pdf main.tex
```

This process is being tested in parallel with the DigitalOcean instance to confirm compatibility between both platforms.

9.9 Step 7: Version Control and Hash Key (In Progress)

To map each document version to a Git commit, versioning will be added to the LaTeX title once the sync process is finalized:

```
\title{SSW 590 - Domain Names, SSL, and Versioning (v1.0 - Commit 6fdbbf1)}
```

This ensures full traceability between the Overleaf PDF and the GitHub repository version.

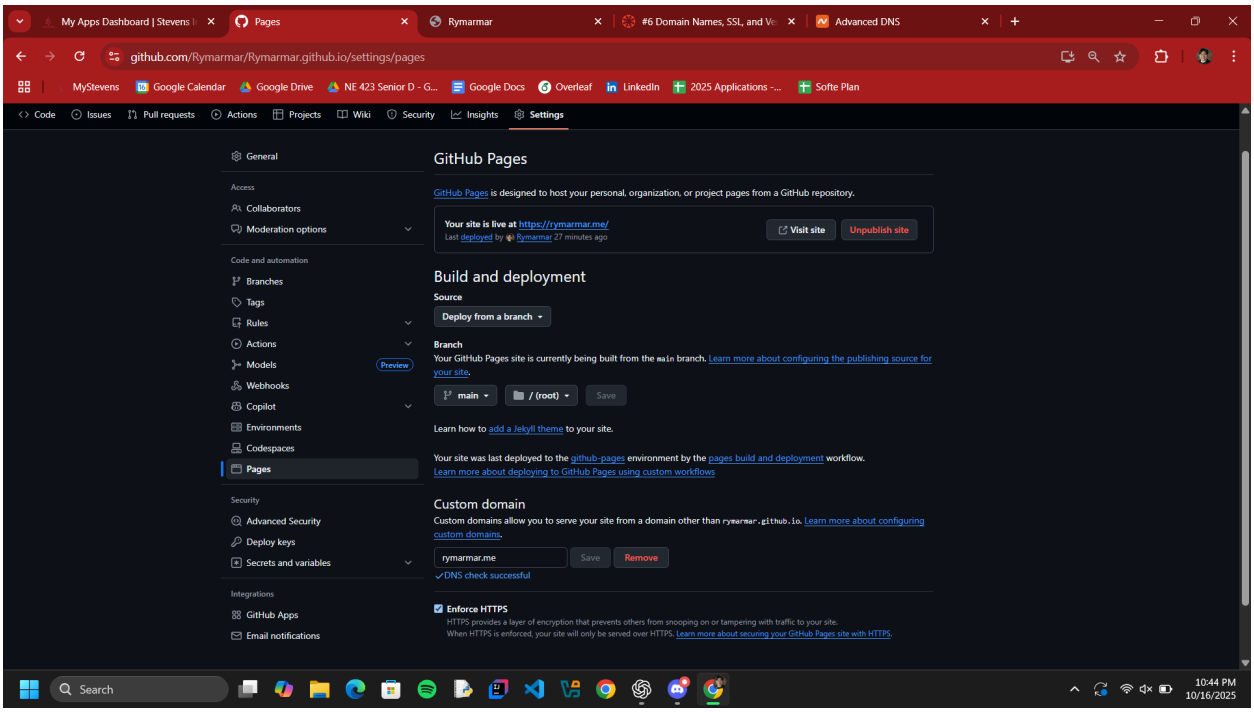

Figure 9.11: GitHub Pages settings confirming DNS validation and HTTPS enforcement.

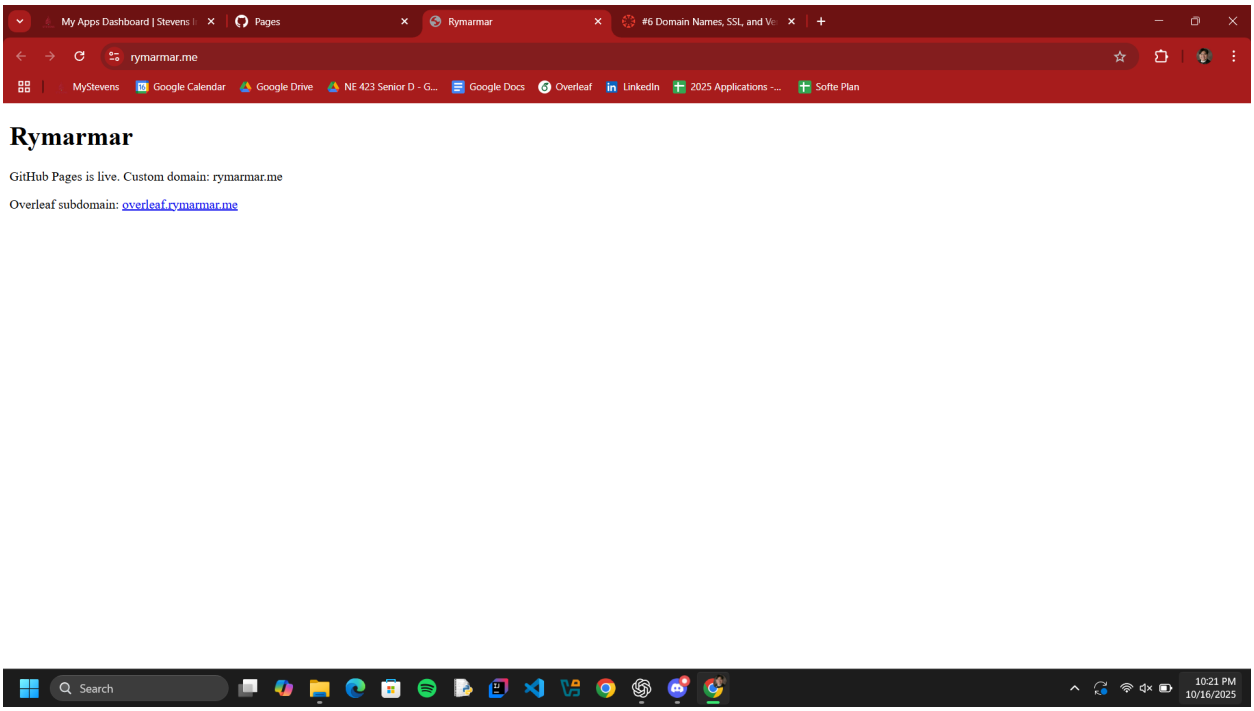

Figure 9.12: Deployed GitHub Pages site confirming build and domain resolution for `rymarmar.me`.

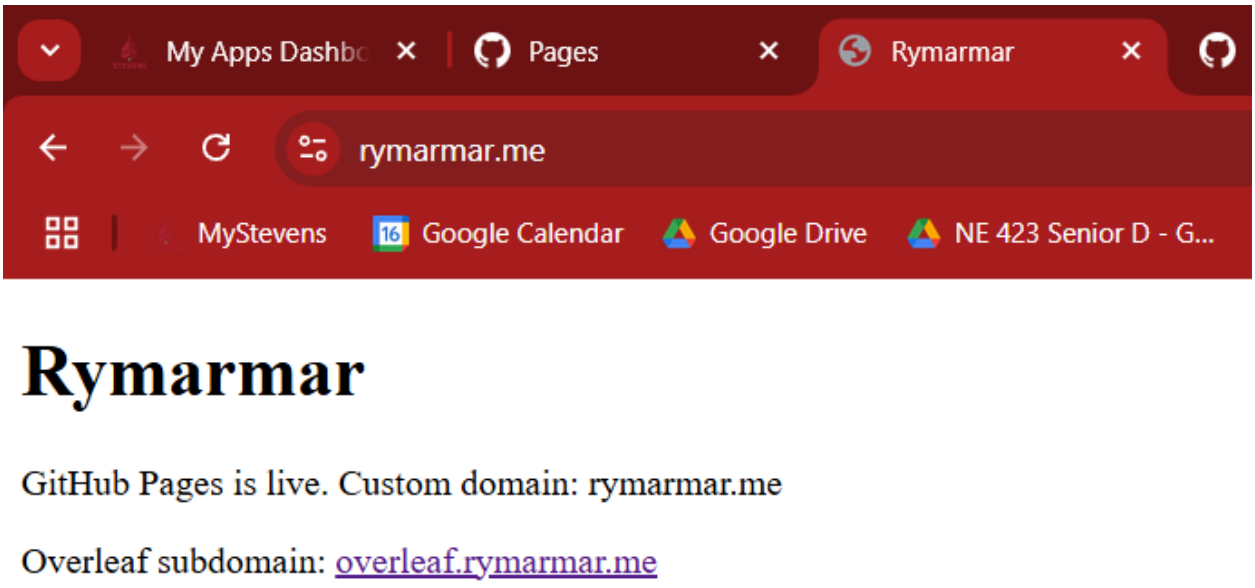

Figure 9.13: The subdomain `overleaf.rymarmar.me` redirecting to the main site `rymarmar.me`.

9.10 Conclusion and Deliverable Checklist

This assignment demonstrates the end-to-end setup of a secured domain with SSL, integration with GitHub Pages, and configuration for an Overleaf instance hosted on DigitalOcean. Minor remaining work includes automating Overleaf <-> GitHub syncing and embedding commit hashes in titles for traceability.

1. **Domain Name:** Registered `rymarmar.me` via GitHub Student Pack – ✓
2. **SSL Configuration:** HTTPS enforced via GitHub Pages / Let's Encrypt – ✓
3. **Overleaf Container Setup:** Hosted on DigitalOcean droplet – ✓
4. **GitHub Pages Integration:** Site deployed and DNS verified – ✓
5. **Overleaf Subdomain Redirect:** Verified redirect to GitHub Pages – ✓
6. **Overleaf <-> GitHub Sync:** Manual workflow setup – *In Progress*
7. **Version/Hash in Title:** Format added, implementation pending – *In Progress*

Appendix A

Appendix
– Author Name

Bibliography

Index