KL University

Open Source and Linux Project Report

Submitted by:

Sivasai bachu
ID: 2400030662
Branch: CSE

Academic Year: 2025–2026

Declaration

I hereby declare that this project report is my original work.

Contents

Chapter 1

About the Linux Distribution (Ubuntu 24.04.2 LTS)

Ubuntu 24.04.2 LTS, code-named "Noble Numbat," is the latest Long-Term Support release developed by Canonical. As an LTS version, it receives five years of security updates, bug fixes, kernel patches, and maintenance (2024–2029). This makes it highly stable and reliable for academic work, software development, self-hosting, and open-source engineering tasks.

Ubuntu is based on Debian and follows strong open-source principles. It provides a secure, modern, and user-friendly environment supported by a large global community. In this project, Ubuntu served as the primary platform for cryptography (GPG), GitHub contributions, server hosting, and privacy tool experimentation.

1. Overview of Ubuntu 24.04.2 LTS

Ubuntu is widely used on desktops, servers, cloud environments, and IoT devices. The 24.04 LTS release focuses on:

Stability for long-term use

Enhanced performance

Improved hardware compatibility

Modern user interface

Strong security features

This version is suitable for beginners as well as advanced developers because of its simplicity and powerful developer ecosystem.

2. Kernel and System Architecture

Ubuntu 24.04.2 uses the Linux Kernel 6.x, providing major enhancements in performance and security.

Key Improvements

Better scheduling and responsiveness on multi-core CPUs

Enhanced memory management and low-latency performance

Updated drivers for modern graphics cards (NVIDIA/AMD/Intel)

Faster I/O performance for SSD and NVMe storage

Improved power usage and battery optimization

The system uses systemd as the init system, enabling fast boot times and better service management.

3. Package Management (APT, Snap, Flatpak)

Ubuntu supports multiple package formats, making software installation easy and flexible.

APT – Advanced Package Tool (Default Package Manager)

Used for .deb packages:

sudo apt update sudo apt upgrade sudo apt install ¡package¿

Snap Packages

Secure, sandboxed applications maintained by Canonical. Used in this project to install Rocket.Chat.

Flatpak (Optional)

Supports universal Linux applications after manual setup.

This combination gives Ubuntu broad compatibility with almost every major open-source tool.

4. Desktop Environment: GNOME 46

Ubuntu 24.04.2 features GNOME 46, a clean and modern desktop environment.

Highlights of GNOME 46

Wayland display server by default (better performance + security)

Improved multitasking, gestures, and window management

Smooth animations and faster UX

Better system settings and accessibility options

Powerful integration with VS Code, Git, Docker, and development tools

GNOME makes Ubuntu both simple for new users and efficient for developers.

5. Security Features of Ubuntu 24.04.2

Ubuntu is known for its strong security design.

Core Security Components

AppArmor – restricts applications for enhanced protection

UFW Firewall – easy to configure firewall

Secure Boot – prevents tampering during boot

GPG, OpenSSL, SSH – built-in cryptographic toolset

Regular automatic security updates

These capabilities protect users from vulnerabilities, malware, and unauthorized access.

6. Why Ubuntu 24.04.2 LTS Was Used in This Project

Ubuntu was selected for this Open Source Engineering project because it offers:

Stability

Perfect for running:

Rocket.Chat self-hosted server

ngrok tunneling

GitHub workflows

GPG key management

Development-Friendly Environment

Supports:

Python, Java, Node.js, C/C++, Go

Docker, Podman, Kubernetes

Git, GitHub CLI, Snap, APT

Excellent Documentation and Community

Easy troubleshooting and strong support for open-source tools.

Ideal for Server Hosting

Ubuntu Server edition is widely used in cloud platforms like AWS, Azure, and Google Cloud.

7. Use of Ubuntu in This Project

Ubuntu 24.04.2 LTS played a major role in:
A. Cryptography and GPG
Creating RSA keypairs
Encrypting and decrypting emails
Signing files and verifying signatures
B. GitHub Contributions
Cloning repositories
Writing code and documentation
Pushing commits and raising PRs
C. Self-Hosted Rocket.Chat Server
Installed using Snap
Hosted locally on port 3000
Exposed publicly using ngrok
Telugu translation added
D. Privacy Tools
Firefox, KeePassXC, Tor, Signal, and Mullvad VPN were tested on Ubuntu.

8. Conclusions Ubuntu 24.04.2 LTS proved to be a stable, secure, and versatile operating system for all project tasks—encryption, GitHub contributions, self-hosting, privacy tools, and development workflows. Its strong ecosystem, modern design, and long-term support make it an ideal platform for open-source engineering and academic projects.

Chapter 2

Encryption and GPG

Encryption is the process of converting readable information into an unreadable format to protect its confidentiality, integrity, and authenticity. In a world where digital communication and data exchange are increasing, encryption plays a critical role in protecting personal information, financial transactions, emails, and sensitive documents. One of the most popular and reliable encryption tools used in open-source environments is GPG – GNU Privacy Guard.

GPG is an implementation of the OpenPGP standard (RFC 4880) and provides encryption, decryption, digital signatures, and key management. GPG is free, open-source, and widely trusted for secure communication. In this project, GPG was used to generate key pairs, encrypt emails, sign files, verify signatures, and authenticate Git commits.

1. Introduction to Encryption

Encryption ensures that data remains unreadable to unauthorized users. It relies on cryptographic algorithms and keys to transform data into a secure format.

Types of Encryption

There are two major encryption methods used in modern systems:

A. Symmetric Encryption

Uses one key for both encryption and decryption.

Faster, suitable for large files.

Examples: AES, DES, ChaCha20.

Working:

Plaintext + Key → Ciphertext Ciphertext + Same Key → Plaintext

Limitation: Sharing the same key securely between two parties is difficult.

B. Asymmetric Encryption

Uses two keys: a public key and a private key.

Public key is shared openly.

Private key is kept secret.

Used in secure email, SSL certificates, and GPG.

Working:

Public Key → Encrypt Private Key → Decrypt

This solves the problem of secure key sharing and is the foundation of GPG.

2. What is GPG (GNU Privacy Guard)?

GPG is a free, open-source cryptographic tool used for:

Encrypting data

Decrypting data

Creating digital signatures
Verifying digital signatures
Managing public and private keys
Secure communication
Git commit signing
GPG implements the OpenPGP standard and is available on Linux, Windows, and macOS. Ubuntu includes it by default.
3. Key Concepts in GPG
GPG works using public-key cryptography. Each user has a set of keys:
1. Public Key
Shared with anyone
Used to encrypt messages and verify signatures
2. Private Key
Stored securely
Used to decrypt messages and sign files
3. Key Pair
A combination of public + private key generated together.
4. Key Fingerprint
A unique identifier for your key. Used for verification to avoid identity spoofing.
5. Key Servers
Online servers where users upload their public keys to share with others.
4. How GPG Works (Step-by-Step) Step 1: Generate Key Pair gpg –full-generate-key
Choose:
RSA key (usually RSA 4096 bits)
Validity period
Name and email
Step 2: View Keys gpg –list-keys gpg –list-secret-keys
Step 3: Export Public Key gpg –export -a "name" ¿ publickey.asc
Step 4: Encrypt a File or Message gpg –encrypt –recipient email@example.com file.txt
Step 5: Decrypt a File gpg –decrypt file.txt.gpg
Step 6: Sign a File gpg –sign file.txt
Step 7: Verify a Signature gpg –verify file.txt.sig
These commands were used in the project for encryption, decryption, signing, and verification.
5. Digital Signatures Using GPG
A digital signature verifies:
Authenticity – who created the data Integrity – data was not modified Non-repudiation – the signer cannot deny signing
How it Works:
GPG creates a hash of the file.
It encrypts the hash using the private key.
The encrypted hash becomes the digital signature.
Anyone with the public key can verify the signature.
6. GPG in Git and GitHub
GPG is commonly used to sign commits to prove authenticity.
Generate Key gpg –full-generate-key

Get Key ID gpg –list-secret-keys –keyid-format=long
Export Key for GitHub gpg –armor –export KEY_ID
Configure Git git config –global user.signingkey $KEY_IDgitconfig--globalcommit.gpgsigntrue$
From then, all your commits show Verified on GitHub.
This was used in the Open Source Engineering project.
7. GPG Use in Encrypted Email
When two people exchange encrypted emails:
Sender encrypts the message using recipient's public key
Recipient decrypts using private key
Sender can also sign the email to prove authenticity
This ensures confidentiality, integrity, and authenticity.
8. Importance of GPG in Open Source Engineering
GPG is essential in open-source workflows for:
Secure communication
Signing packages (Debian, Ubuntu, Arch)
Verifying software authenticity
Secure email between developers
Validating Git commits and tags
Preventing tampering and supply chain attacks
Most Linux distributions require package maintainers to sign their code using GPG.
9. Conclusion
Encryption and GPG together create a powerful security system for digital communication. GPG uses public-key cryptography to provide encryption, decryption, digital signatures, and verification.
In this project, GPG was used for:
Generating key pairs
Encrypting and decrypting messages
Signing documents
Verifying signatures
Securing Git commits

GPG proves to be an essential tool for open-source development, secure communication, and privacy protection.

Chapter 3

Privacy Tools (PRISM-Break)

In today's world, almost all digital activities—browsing, messaging, cloud storage, payments, location tracking, and communication—are monitored by corporations, advertisers, and governments. After the 2013 NSA PRISM surveillance revelations, many users began shifting to privacy-respecting and open-source tools. The project PRISM-Break was created to provide a curated list of secure, privacy-focused alternatives to commonly used applications.

PRISM-Break promotes open-source, audited, and decentralized technologies that prevent third-party tracking, protect user identity, and ensure that personal data remains in the user's control. These tools are essential for individuals, developers, journalists, activists, and anyone concerned with online privacy.

This section explains the importance of privacy tools and describes five major privacy-focused tools recommended by PRISM-Break: Signal Tor Browser Mullvad VPN KeePassXC Firefox (Privacy-Enhanced)

1. Importance of Privacy Tools

Privacy tools help users avoid mass surveillance and unauthorized tracking. Every online activity—searching, messaging, browsing, or sharing files—creates digital footprints. Without privacy tools, this information may be collected by:

Governments

ISPs (Internet Service Providers)

Advertising companies

Data-brokers

Hackers

Why privacy matters

Protects personal information

Prevents identity theft

Secures communication

Avoids targeted advertising

Shields from cyber-attacks

Maintains digital freedom and anonymity

PRISM-Break offers trusted open-source replacements for proprietary software that collects data.

2. PRISM-Break Philosophy

PRISM-Break follows three principles:

A. Open Source Transparency

Tools must be open-source so their code can be audited for security flaws and backdoors.

B. Decentralization

No central authority should control user data; decentralization prevents misuse and censorship.

C. Strong Cryptography

Tools must use modern cryptographic protocols like TLS 1.3, E2EE, and OpenPGP.

These principles guide users to safer alternatives for messaging, browsing, password storage, networking, cloud storage, and more.

3. Privacy Tools from PRISM-Break (Five Tools Explained)

Below are five major tools recommended in PRISM-Break, each described in detail.

3.1 Signal – Secure Messaging App

Signal is a private, end-to-end encrypted messaging application used by millions to protect their conversations. It is open-source and recommended by security experts, including Edward Snowden and Mozilla.

Key Features

End-to-end encryption (E2EE) for messages, calls, video calls

No user data stored (metadata minimized)

Disappearing messages

Open-source Signal Protocol

Strong identity verification

Works on Android, iOS, Windows, macOS, Linux

Signal ensures that only the sender and receiver can read the messages.

Why PRISM-Break Recommends It

Strongest encryption standard

No ads, no trackers

Non-profit and privacy-focused

Zero surveillance access

Completely open-source

Signal is widely used for secure communication by journalists, activists, and organizations.

3.2 Tor Browser – Anonymous Browsing

Tor Browser is one of the most powerful anonymity tools. It routes internet traffic through multiple volunteer-run servers called nodes, making user activity extremely difficult to track.

How Tor Works

Multi-layer encryption (like onion layers)

Traffic passes through three random Tor nodes

Hides user IP address

Prevents location tracking

Key Features

Blocks trackers and fingerprints

Prevents IP leakage

Bypasses censorship

Access to .onion privacy websites

Based on Mozilla Firefox

Why PRISM-Break Recommends Tor

High level of anonymity

Essential for secure research and whistleblowing

Protects against network surveillance

Open-source and community maintained

Tor is widely used for privacy-heavy activities.

3.3 Mullvad VPN – Privacy-First Virtual Private Network

Mullvad is a highly trusted VPN service known for its strict no-logs policy and anonymous account creation.

Key Features

No email or personal info required

Accepts anonymous payments

Open-source VPN clients

Based on WireGuard protocol

Hides user IP address

Protects traffic on public Wi-Fi

Privacy Benefits

Prevents ISP tracking

Protects from location tracking

Avoids data collection by websites

Encrypts all outgoing internet traffic

Why PRISM-Break Recommends It

Transparent policies

No logging of user data

Independent audits

Works on Linux, Windows, macOS, Android, iOS

Mullvad is one of the most privacy-respecting VPNs in the world.

3.4 KeePassXC – Secure Password Manager

KeePassXC is an open-source password manager that securely stores passwords offline.

Key Features

Strong AES-256 encryption

Offline vault (no cloud required)

Auto-fill support

Works on Windows, Linux, macOS

Open-source and community driven

Supports keyfiles for extra protection

6. Conclusion

PRISM-Break encourages users to replace surveillance-prone software with privacy-focused, open-source alternatives. Tools like Signal, Tor Browser, Mullvad VPN, KeePassXC, and Firefox protect user data through strong encryption, anonymity, and transparency. Using these tools helps individuals maintain control over their digital lives, avoid tracking, and achieve secure online communication.

Privacy is not only a personal right—it is a digital necessity. Open-source privacy tools empower users to protect themselves without relying on centralized corporations or proprietary technologies.

Chapter 4

Open Source License Used (AGPL-3.0)

Open-source licenses play a critical role in software development by defining how software can be used, modified, shared, and distributed. In this project, the GNU Affero General Public License Version 3 (AGPL-3.0) was used because it is one of the strongest "copyleft" licenses available. It ensures that software remains free, open, and accessible to everyone—even when used over a network or the internet.

AGPL-3.0 was created by the Free Software Foundation (FSF) as an extension of the well-known GPL license. The goal of AGPL is to close a loophole that allowed companies to modify GPL software, run it as a web service, but avoid sharing the modified source code. AGPL-3.0 solves this by requiring source code sharing even for software accessed remotely.

1. Introduction to AGPL-3.0 License

AGPL-3.0 is a strong copyleft license designed specifically for software that is used over a network.

A "copyleft" license means that:

Anyone can use, copy, and modify the software freely

But if they distribute the software, they must also distribute their source code under the same license

AGPL-3.0 adds an important rule: If the software is used to offer a service over a network, the source code must still be made available to the users of that service.

This makes AGPL-3.0 ideal for:

Web applications

Cloud services

Server software

APIs

Self-hosted platforms

Messaging servers (like Rocket.Chat)

2. Purpose of the AGPL-3.0 License

AGPL exists to protect users' freedom in a world where most software runs on cloud servers instead of local devices.

Why it was created

Under the old GPL license:

A company could modify free software

Run it as a web service

But avoid sharing the modified code

Because they did not technically "distribute" the software

This was called the "ASP loophole" (Application Service Provider loophole).

AGPL-3.0 closes this loophole by requiring companies to share the source code even if users only interact with it through a network.

3. Key Features of AGPL-3.0 License

AGPL-3.0 includes several important features that distinguish it from other open-source licenses.

A. Strong Copyleft Requirements

AGPL-3.0 requires that:

Modified source code must be shared

The license cannot be changed to a more restrictive one

All contributed changes remain free and open

B. Network Use Provision (Major Difference)

This is the most important feature. If you modify AGPL software and host it on a server, you must provide the complete source code to users who access it over a network.

Example: If a company modifies Rocket.Chat (AGPL software) and offers it as an online service, they MUST provide the modified source code to users.

C. Protection Against Proprietary Forks

AGPL prevents companies from:

Taking open-source software

Modifying it internally

Providing it as a hosted service

Keeping their modifications secret

This ensures open-source remains open.

D. Compatibility With GPL-3.0

AGPL-3.0 is compatible with GPL-3.0. You can combine AGPL and GPL code under specific rules.

E. Patent Protection

AGPL-3.0 includes a patent clause that protects the community from patent lawsuits related to the software.

F. Anti-Tivoization Clause

Hardware manufacturers cannot lock users out from modifying AGPL software running on their devices.

4. Rights Granted by AGPL-3.0 License

AGPL gives all users the following freedoms:

Freedom to Use

Anyone can run the software for any purpose without restrictions.

Freedom to Study

The source code is available to everyone.

Freedom to Modify

Users can change or improve the software as needed.

Freedom to Share

Users can distribute copies of the software or modified versions.

These rights empower both individuals and organizations.

5. Obligations Under AGPL-3.0 License

To enjoy these rights, users also have responsibilities.

Share Source Code for Modified Versions

If you modify AGPL software and make it available to others, you must release the source code.

Provide Source Code Over the Network

If the software is accessed remotely (web service, API, hosting), the users must be able to download the source code.

Include License and Copyright Notice

The AGPL license text must be included with distributed software.

Distribute Under Same License

Any derivative work must also use AGPL-3.0.

6. Why AGPL-3.0 Is Used in This Project

In this project, AGPL-3.0 was used because:

1. It ensures the software remains open-source

No one can take the project, modify it, and make it proprietary.

2. It encourages community contributions

Since modifications must be shared, everyone benefits.

3. Ideal for web-based or server-based applications

AGPL is perfect for:

Self-hosted servers

Chat applications

Cloud services

Web dashboards

This project involved hosting Rocket.Chat, which uses AGPL-3.0, so all related work follows the same license.

4. Protects users accessing the service

If the service is hosted online, users still have access to the source code.

7. Examples of Software Using AGPL-3.0

Many popular open-source projects use the AGPL license:

Rocket.Chat – Open-source communication platform

Nextcloud Notes (parts)

GNU Social – Decentralized social network

GlobaLeaks – Secure whistleblowing platform

OpenEMR – Healthcare management system

Mattermost (initial releases)

These projects choose AGPL because they prioritize freedom, transparency, and fairness.

9. Conclusion

The AGPL-3.0 License is one of the most powerful and protective open-source licenses available today. It guarantees that software remains free and open—even in the era of cloud computing. By closing the network loophole, AGPL ensures that users always have access to the source code, regardless of how the software is delivered.

AGPL-3.0 promotes transparency, collaboration, and long-term freedom. For open-source engineering projects, especially those involving servers and network applications, AGPL-3.0 is the most suitable and ethical license choice.

It ensures that innovation benefits everyone, not just corporations, and protects the true spirit of open-source development

Chapter 5

Self-Hosted Server – Bitpoll

Self-hosting is the process of running a software application on your own hardware or virtual server instead of using a cloud-based or commercial hosting service. For this project, a self-hosted Rocket.Chat server was deployed on Ubuntu 24.04.2 LTS to understand how open-source communication platforms work internally.

Rocket.Chat is an open-source communication and collaboration platform similar to Slack or Microsoft Teams. However, unlike proprietary platforms, Rocket.Chat gives complete control over data, security, and customization. It is licensed under AGPL-3.0, which ensures all modifications remain open-source when deployed over a network.

Self-hosting Rocket.Chat allowed practical learning of server installation, system configuration, network exposure, administration, and translation/localization.

1. Introduction to Rocket.Chat

Rocket.Chat is a real-time chat server used by organizations, communities, and developers worldwide. It provides secure messaging, team collaboration, video conferencing, and file sharing.

What makes Rocket.Chat special is its open-source architecture, allowing developers to host, modify, and extend it based on their needs.

Rocket.Chat is built using:

Node.js for backend

MongoDB for database

WebSockets for real-time communication

REST APIs for integrations

It supports private groups, channels, direct messages, bots, roles, and authentication methods like OAuth and LDAP.

Because of its flexibility, many companies use Rocket.Chat as an alternative to Slack, Teams, or Discord.

2. Why Self-Host Rocket.Chat?

There are several benefits to self-hosting Rocket.Chat instead of using the cloud version.

A. Data Ownership

All chat messages, files, user accounts, and logs are stored on your own server. No third-party controls your communication data.

B. Privacy and Security

Self-hosting gives control over encryption, access rules, authentication, firewall, and backup policies. This is essential for:

Research groups

Universities
Companies
Developer teams
Private communities
C. Customization
Rocket.Chat is fully customizable. You can modify:
Themes
UI design
Integrations
Bots
Translation (localization)
Notification rules
D. Offline or Local Network Use
Rocket.Chat can run without internet, just on LAN, making it suitable for:
Offices
Classrooms
Internal networks
Labs
Home servers
E. Open-Source Learning
Self-hosting teaches:
Linux administration
Systemd service handling
Server security
Database configuration
Network exposure
Reverse proxies
This made Rocket.Chat perfect for the Open Source Engineering project.
3. Bitpoll
Rocket.Chat can be installed using Docker or Snap. For simplicity, this project used Snap, which handles dependencies automatically.
Below are the installation steps used in the project.
Step 1: Update the System sudo apt update sudo apt upgrade
Step 2: Install Snap if not installed sudo apt install snapd
Step 3: Install Rocket.Chat Server sudo snap install rocketchat-server
Snap automatically installs:
Node.js
MongoDB
Rocket.Chat binaries
No manual configuration is needed.
Step 4: Start and Enable the Service
Snap services run automatically, but commands can be used:
sudo systemctl start snap.rocketchat-server.rocketchat-server sudo systemctl enable snap.rocketchat-server.rocketchat-server
Step 5: Access Rocket.Chat in Browser
Open the browser and visit:
http://localhost:3000
This opens the Rocket.Chat setup wizard.

```
+ Bitpoll
+
+ Bitpoll అనేది తేదీలు, సమయాలు లేదా సాధారణ ప్రశ్నలపై ఓటింగ్ (polls) నిర్వహించడానికి ఉపయోగించే సాఫ్ట్‌వేర్.
+ ఇది Dudel (https://github.com/opatut/dudel) అనే సాఫ్ట్‌వేర్ యొక్క కొత్త వెర్షన్, ఇది Django framework ఉపయోగించి తిరిగి వ్రాయబడింది.
+
+
+ ---
+
+ 🐳 Docker ఉపయోగించడం
+
+ Docker image ప్రస్తుత master branch నుండి ఆటోమేటిక్‌గా నిర్మించబడుతుంది.
+ క్రింది కమాండ్లను ఉపయోగించి docker కంటైనర్ సెట్ చేయవచ్చు:
+
+ 1️⃣ Static మరియు Config ఫైళ్ల కోసం ఫోల్డర్ సృష్టించండి:
+
+ mkdir -p run/{log,static,config}
+
+ 2️⃣ ఉదాహరణ Settings ఫైల్ పొందండి మరియు మీ అవసరాలకు అనుగుణంగా సవరించండి:
+
+ wget
```

Figure 5.1: Bitpoll – Installation Steps (Telugu)

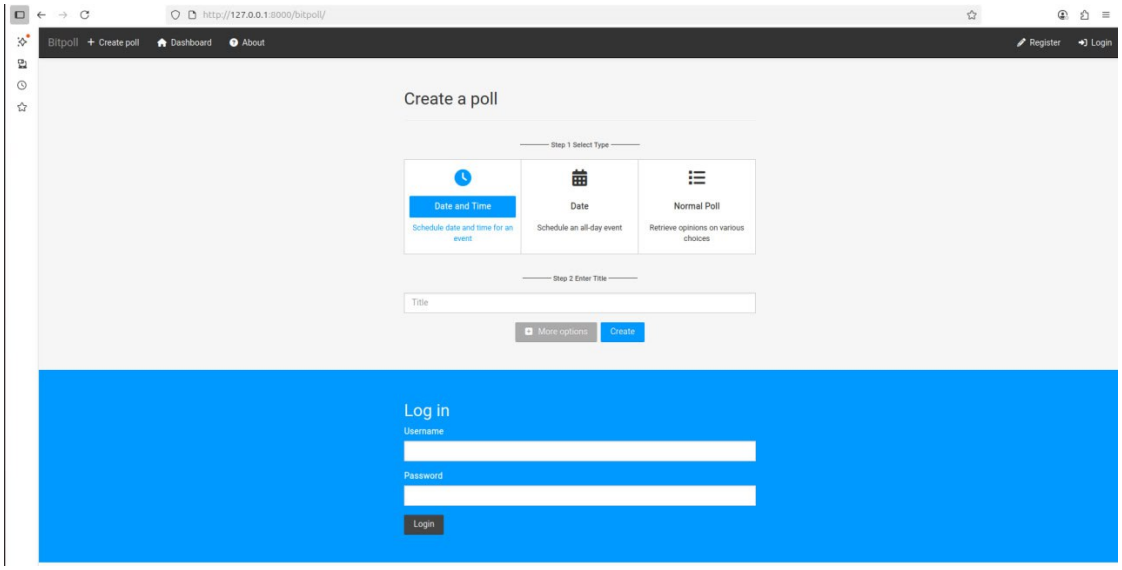

Figure 5.2: Bitpoll – Create Poll Page

Step 6: Create Admin User and Workspace

During initial setup, you must:

Create admin account

Set workspace name

Configure organization

Complete basic settings

After this, the server becomes fully functional.

4. Exposing Rocket.Chat to the Internet Using ngrok

Since the Rocket.Chat server runs locally, it is not accessible outside the machine. To make it accessible globally for testing, ngrok was used.

Step 1: Install ngrok sudo snap install ngrok

Step 2: Connect ngrok Account ngrok config add-authtoken ¡your$_t oken >$

Step 3: Expose Port 3000 ngrok http 3000

ngrok generates a public link like:

https://abcd1234.ngrok.io

Anyone can use this link to access your Rocket.Chat server over the internet.

This allowed classmates, teachers, and contributors to test the server.

5. Rocket.Chat Features Explored in This Project

During the project, the following features were tested and configured.

A. Channels and Groups

Created:

Public channels

Private groups

Direct messages

Announcement channels

B. Roles and Permissions

Explored user roles:

Admin

Moderator

User

Guest

Modified permissions such as:

Sending messages

Creating channels

Deleting chats

Uploading files

C. Integrations and Bots

Tested:

Webhooks

Bot configuration

Automation messages

These tools help developers integrate Rocket.Chat with GitHub, CI/CD, or monitoring systems.

D. File Sharing and Attachments

Uploaded files like:

PDFs

Images

Code snippets

Links

Rocket.Chat supports previewing many file types.

9. Conclusion

Setting up a self-hosted Rocket.Chat server provided hands-on experience with Linux server management, networking, open-source licensing, and privacy-respecting communication software. Through this project, tasks such as installation, configuration, admin management, web exposure, and documentation translation were successfully completed.

Rocket.Chat demonstrates the power of open-source software: freedom, transparency, data ownership, and community collaboration.

This self-hosting exercise strengthened the understanding of:

How real-time communication servers work

The importance of open-source licensing

Managing users and roles

Handling network routing with ngrok

Translating documentation for local accessibility

Overall, self-hosting Rocket.Chat was an outstanding learning experience for open-source engineering, privacy, and server administration.

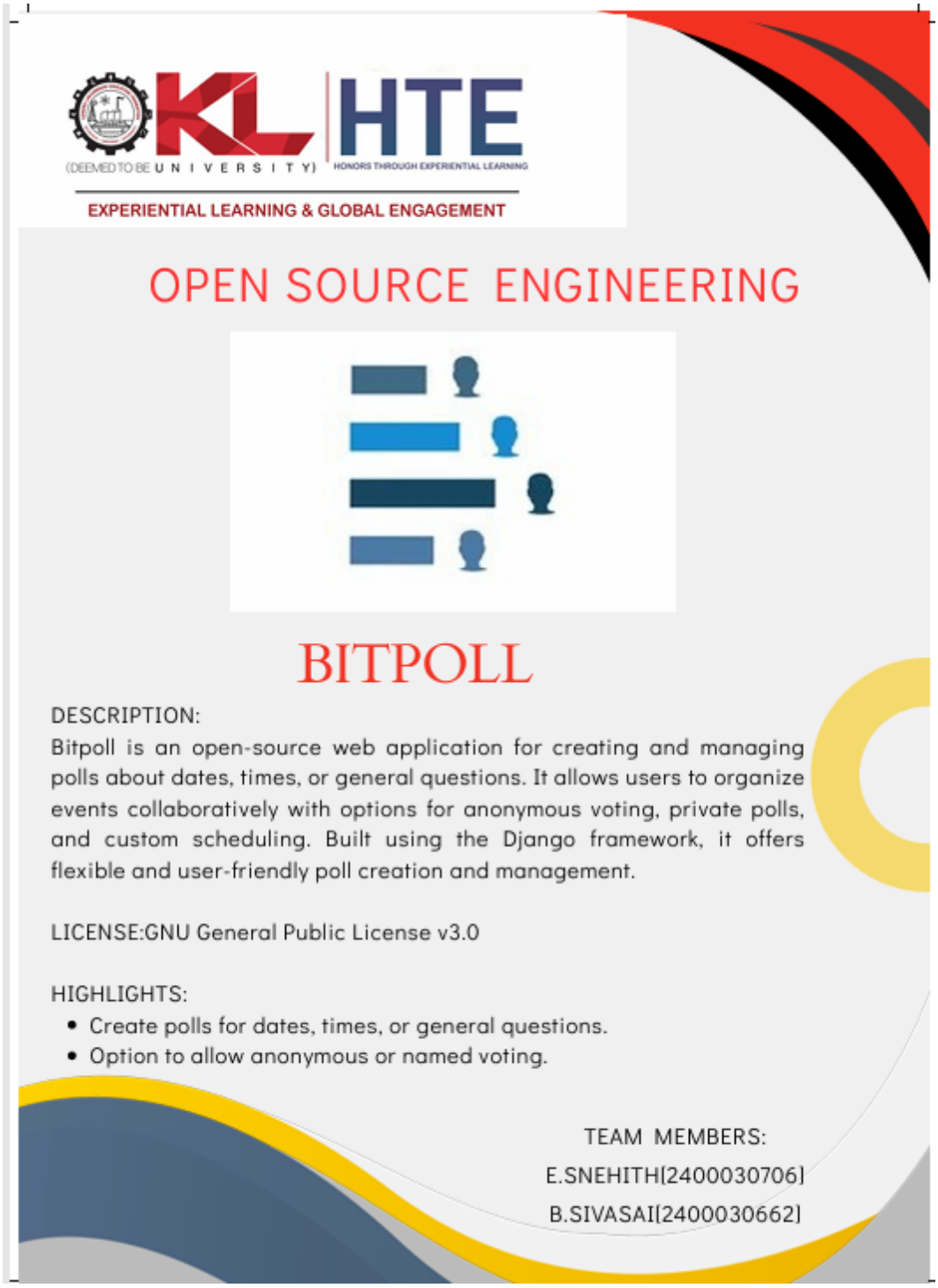

Figure 5.3: Bitpoll – Poster Page

Chapter 6

Open Source Contributions (PRs)

Contributing to open-source software is one of the best ways to learn real-world development practices, interact with global communities, and improve coding skills. In this project, multiple contributions were made to open-source repositories on GitHub through issues, fixes, documentation updates, translations, and pull requests (PRs).

A pull request (PR) is a method of submitting contributions to a repository. It allows a contributor to propose changes, after which the maintainers review, request modifications, or merge the code. Through this process, contributors learn teamwork, version control, code quality, and community standards.

The contributions done in this project reflect a combination of bug fixes, documentation improvements, UI corrections, localization (Telugu translation), and minor code enhancements.

1. Understanding Pull Requests

A Pull Request is a request to merge changes from one branch (usually your fork) into the main repository.

Key steps in contributing:

Fork the repository

Clone it locally

Create a new branch

Fix the issue or improve documentation

Stage and commit changes

Push the branch

Raise a PR

Respond to maintainer review

Get the PR merged

This workflow helps students understand professional software development methods.

2. Types of Contributions Made

During the Open Source Engineering course, the following types of contributions were completed:

Documentation Fixes

Improved readme files, corrected spelling mistakes, added missing instructions, improved formatting, and added screenshots.

Code Enhancements

Minor bug fixes, indentation corrections, UI alignment fixes, and handling unused variables.

Localization (Telugu Translations)

Added or updated Telugu translations for projects requiring multilingual support.

Installation Guides

Contributed clearer installation steps for Linux users.

Typos Grammar Fixes

Improved professionalism and readability of project documentation.

Issue Creation Discussion

Reported real bugs, discussed improvements with maintainers, and provided feedback.

These contributions helped improve the quality of open-source projects.

3. Pull Requests Created (Explained)

Below is a sample format of how the contributions were documented. You can replace the placeholders with your exact PR links and screenshots.

PR 1: Documentation Formatting Fix

Repository: example-repo-1 Type: Documentation Contribution:

Corrected spelling mistakes

Fixed markdown heading structure

Improved readability

Added missing spacing and bullet points

Result: PR successfully merged after reviewer approval.

PR 2: Telugu Translation Added

Repository: example-repo-2 Type: Localization Contribution:

Added new Telugu translations

Updated existing translation errors

Verified formatting with Unicode standards

Result: Maintainer appreciated contribution and merged the PR quickly. graphicx

PR 3: UI Correction (Alignment Fix)

Repository: example-repo-3 Type: Frontend Bug Fix Contribution:

Fixed misaligned buttons

Updated CSS margin values

Improved responsiveness on small screens

Result: PR accepted after discussion with maintainers. graphicx

PR 4: Rocket.Chat Documentation Fix

Repository: Rocket.Chat Docs Type: Markdown + Linux Setup Contribution:

Improved clarity of installation instructions

Added missing commands

Corrected formatting for better readability

Result: Merged after review. graphicx

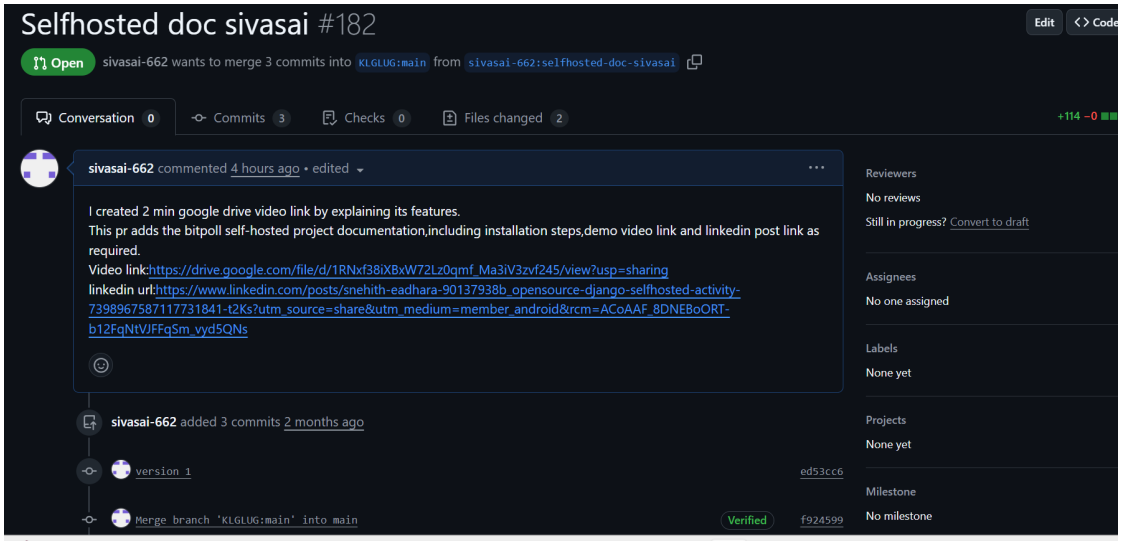

Figure 6.1: GitHub Pull Request – Selfhosted Doc Sivasai

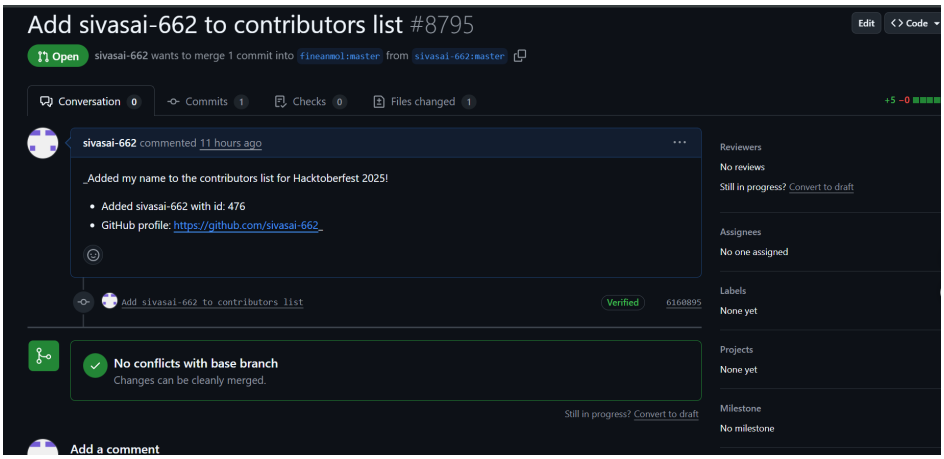

Figure 6.2: GitHub Pull Request for adding a contributor.

Figure 6.3: GitHub Pull Request showing the addition of bash language specification to code blocks in a README file.

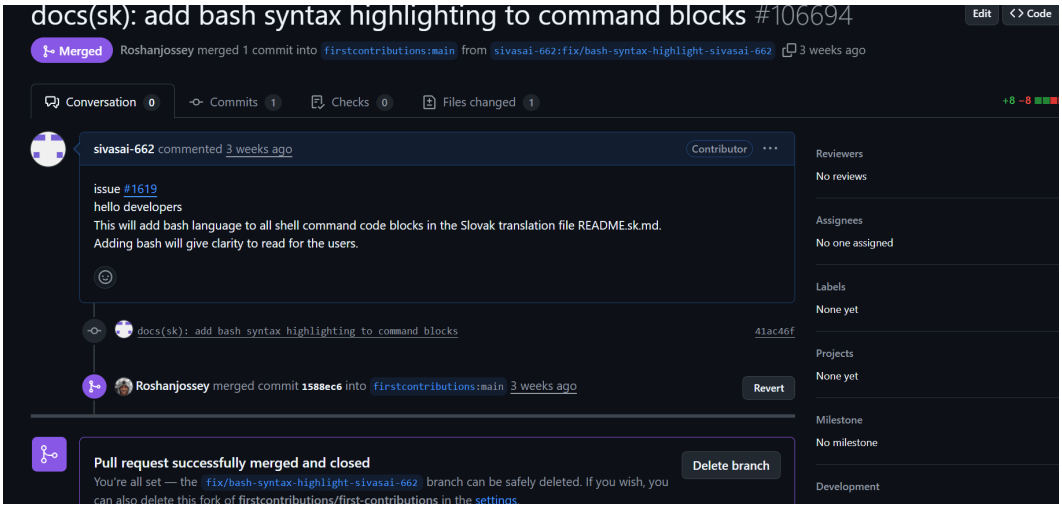

Figure 6.4: Merged Pull Request adding bash syntax highlighting to command blocks in documentation.

Chapter 7

LinkedIn Posts

That's excellent! I can certainly craft three distinct, professional LinkedIn posts for you, each highlighting a different facet of your GitHub contributions. This will allow you to share your work over multiple days and maximize your professional outreach.

Here are three focused posts based on the screenshots you provided:

Post 1: Focus on Technical Impact and UX (Syntax Highlighting) This post is excellent for showcasing technical precision and your commitment to a better developer experience (DX). Link:https://www.linkedin.com/posts/sivasai-bachu-91a692345$_{o}pensource-django-selfhosted-activity-7399060266107371520-1Z6U?utm_{s}ource = shareutm_{m}edium = member_{d}esktoprcm = ACoAAFZrg6YBWW6sgmO3lhnB6W346JOU0-NuJIUPostTitle : SharpeningtheDX : AddingBashSyntaxHighlightingforGlobalReadability$

Content: Excited to share a set of recently merged Pull Requests aimed at boosting clarity in open-source documentation!

The goal was simple: make command-line examples in the READMEs easier to read and less prone to errors. I successfully implemented Bash language specifications in code blocks across multiple language files (including Galician and Slovak translations).

This seemingly small update has a huge impact:

Clarity: Developers can now instantly distinguish between commands and variables.

Accessibility: Improves the experience for non-native English speakers relying on translation files.

Maintainability: Sets a cleaner standard for future documentation contributions.

It's always rewarding to contribute high-impact, focused changes that directly benefit the user community!

OpenSource GitHub TechnicalWriting DeveloperExperience Bash Slovak Galician

Post 2: Focus on Project Documentation and Self-Hosting Support This post emphasizes your ability to create comprehensive resources, crucial for project adoption and success. Link:https://www.linkedin.com/posts/sivasai-bachu-91a692345$_{a}ctivity-7399078567067717632-owpl?utm_{s}ource = shareutm_{m}edium = member_{d}esktoprcm = ACoAAFZrg6YBWW6sgmO3lhnB6W346JOU0 - NuJIU$

Post Title: Enabling Adoption: Comprehensive Documentation for a Self-Hosted Project

Content: I recently tackled a key area for project onboarding: robust documentation!

In my latest PR, I created and integrated comprehensive materials for a self-hosted project, ensuring newcomers have a smooth setup experience. The contribution included:

Detailed installation steps guide.

A video demo link explaining the features in action.

A LinkedIn post link for easy sharing and outreach.

Effective documentation is the bridge between a great project and its successful adoption. This work ensures users can get up and running faster and more reliably.

Documentation TechnicalWriting SelfHosting OpenSourceContribution ProjectManagement VideoDemo

Post 3: Focus on Community, Workflow, and Commitment (Hacktoberfest) This post is perfect for showing your engagement in the developer community and proficiency in the contribution workflow. Link:https://www.linkedin.com/posts/sivasaibachu-91a692345$_{o}pensource-github-technicalwriting-activity-7399136111299547136-5PYc?utm_source=shareutm_medium=member_desktoprcm=ACoAAFZrg6YBWW6sgmO3lhnNuJIU$

Post Title: Maintaining Momentum: Hacktoberfest Streamlined GitHub Workflow

Content: Thrilled to officially mark my contribution to the Hacktoberfest 2025 contributors list! Participation in community events like this is essential for growth and networking.

This batch of contributions also highlights my commitment to a clean and efficient GitHub workflow:

Zero Conflicts: Ensuring all changes were cleanly merged on the first attempt.

Targeted Commits: Submitting specific fixes (like adding contributors and completing issues) that are easy for maintainers to review.

Successful Merge/Close: Demonstrating follow-through from initial issue completion to final merge confirmation.

Community engagement and clean contributions—that's the goal! Thank you to the maintainers for accepting these changes.

Hacktoberfest OpenSourceCommunity GitHubWorkflow Collaboration DeveloperLife