K L UNIVERSITY

COMPUTER SCIENCE ENGINEERING(HTE) DEPARTMENT

ReportOn

TITLE OF THE PROJECT

SUBMITTED BY:

ID NUMBER	NAME
2400030295	PUVVADA BHASKAR

UNDER THE GUIDANCE OF

DR B . Arunekumar Balasubramanian

Assistant Professor

KL UNIVERSITY

Green fields, Vaddeswaram – 522 502

Guntur Dt., AP, India.

DEPARTMENT OF BASIC ENGINEERING SCIENCES

CERTIFICATE

This is to certify that the project based laboratory repor is submitted by PUVVADA BHASKAR 2400030295 Department of COMPUTER SICENCE ENGINEERING

(HTE) , KL University in partial fulfillment of the requirements for the completion of a report in Open source Engenering (24CS02EF)" course in B Tech III Semester, is a Bonafede record of the work carried out by him/her under my supervision during the academic year 2025 – 2026.

PROJECT SUPERVISOR

N B Arunekumar Balasubramanian

HEAD OF THE DEPARTMENT

Dr Aswin

ACKNOWLEDGEMENTS

It is great pleasure for me to express my gratitude to our honorable President. Sri. Koneru Satyanarayana, for giving the opportunity and platform with facilities in accomplishing the project-based laboratory report.

I express sincere gratitude to our Coordinator and HOD-HTE Dr. Aswin for his leadership and constant motivation provided for the successful completion of our academic semester. I record it as my privilege to deeply thank you for providing us with the efficient faculty and facilities to make our ideas into reality.

I express my sincere thanks to our project supisor N B Arunekumar Balasubramanian encouragement, appreciation, and intellectual zeal which motivated us to venture this project successfully.

Finally, we are pleased to acknowledge our indebtedness to all those who devoted themselves directly orindirectly to making this project a success.

Puvvada Bhaskar- 2400030295

Contents

Linux Distribution Used

Name and Version

For this project, I used **Ubuntu 22.04 LTS (Jammy Jellyfish)**. Ubuntu is a Debian-based Linux distribution developed by Canonical and is widely known for its stability, security, and long-term support.

Why You Chose It

I selected Ubuntu 22.04 LTS due to the following reasons:

- **Long-Term Support (LTS):** Receives security and maintenance updates for 5 years.
- **Beginner-Friendly Interface:** GNOME desktop environment makes it easy to use.
- **Strong Community Support:** Large documentation, forums, and tutorials.
- **Wide Software Availability:** Supports development tools, privacy tools, and open-source applications.
- **Enhanced Security:** Includes UFW firewall, AppArmor, frequent security patches.

System Requirements

The minimum and recommended system requirements for Ubuntu 22.04 LTS are shown in Table 1.

Specification	**Minimum Requirement**	**Recommended**
RAM	4 GB	8 GB
Processor	2 GHz Dual-Core	2+ GHz Multi-Core
Storage	25 GB	50+ GB SSD
Graphics	1024x768 resolution support	GPU with 3D Acceleration

Table 1: System Requirements for Ubuntu 22.04 LTS

Installation Steps

The installation process of Ubuntu 22.04 LTS involved the following steps:

1. Downloaded the Ubuntu ISO from the official website.
2. Created a bootable USB using tools like Rufus or Balena Etcher.
3. Booted the system from the USB drive via BIOS/Boot Menu.
4. Selected "Try or Install Ubuntu" and clicked "Install Ubuntu".
5. Chose keyboard layout and installation type (Normal Installation).
6. Selected the installation drive and created a user account.
7. Waited for installation to complete and restarted the system.
8. Performed initial updates using:

```
sudo apt update && sudo apt upgrade
```

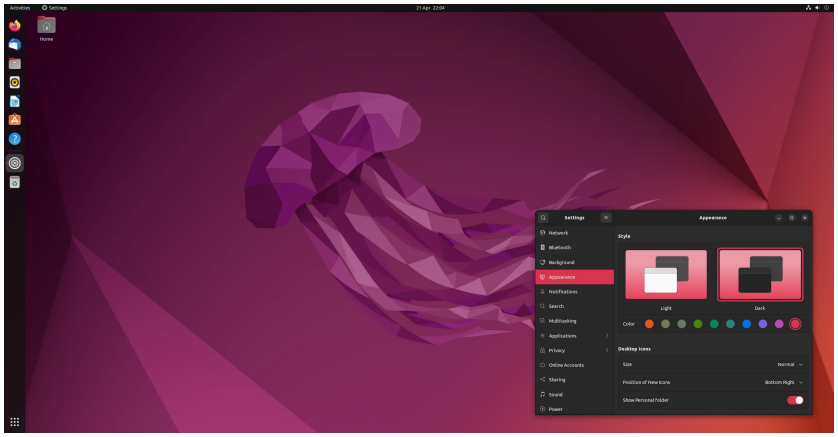

Figure 1: Installation Screenshot

Encryption and GPG

What is Encryption?

Encryption is the process of converting readable data (*plaintext*) into an unreadable form (*ciphertext*) to protect it from unauthorized access. Only someone with the correct key can decrypt and read the data.

There are two major types of encryption:

- **Symmetric Encryption:**
  - Uses a single key for both encryption and decryption.
  - Fast and suitable for encrypting large files.
  - Example algorithms: AES, DES.
  - Limitation: Key must be securely shared between parties.
- **Asymmetric Encryption:**
  - Uses two keys: a **public key** (for encryption) and a **private key** (for decryption).
  - Public key can be shared openly, private key must be kept secret.
  - More secure for communication because keys are not shared.

- Example algorithms: RSA, ECC.

What is GPG?

GNU Privacy Guard (GPG) is an open-source tool that implements asymmetric encryption for secure communication and data protection.

- Based on the OpenPGP standard.
- Allows users to encrypt files, emails, and messages.
- Uses public-private key pairs.
- Supports digital signatures to verify identity and data integrity.
- Works on Linux, including Ubuntu natively.

Generating Keys

To use GPG for encryption, a key pair must be created.

Steps:

1. Open the terminal.
2. Run the command to generate a new key:

```
gpg --full-generate-key
```

3. Select key type (default RSA and RSA).
4. Choose key size (recommended: 4096 bits).
5. Set expiry time.
6. Enter name, email, and password when prompted.

After completion, the public and private keys are stored securely in the GPG keyring.

Import/Export Keys

GPG allows sharing your public key with others and importing their keys for secure communication.

Exporting Public Key

1. List keys:

```
gpg --list-keys
```

2. Export public key:

```
gpg --export -a "Your Name" > publickey.asc
```

3. Share `publickey.asc` with others.

Importing Someone's Public Key

1. Download their public key file.
2. Import it using:

```
gpg --import publickey.asc
```

3. Verify the key fingerprint for authenticity.

Exporting Private Key (Optional)

For backup purposes only:

```
gpg --export-secret-keys -a "Your Name" > privatekey.asc
```

Note: The private key must never be shared.

Sending Encrypted Email

Tools Used

To send encrypted emails using GPG, the following tools were used:

- **Mozilla Thunderbird:** An open-source email client that supports encryption.
- **Thunderbird OpenPGP Built-in Support:** Allows encryption, decryption, and digital signatures without external plugins.
- **GPG (GNU Privacy Guard):** Used for generating and managing public/private key pairs.
- **Email Account (e.g., Gmail):** Used for sending and receiving encrypted mail through Thunderbird.

These tools together ensure secure communication using asymmetric encryption.

Steps

The process of sending an encrypted email involves three main stages: key exchange, encryption, and sending.

1. **Configure Thunderbird**
    - Install and open Mozilla Thunderbird.
    - Add your email account (e.g., Gmail) by signing in.
    - Open Account Settings and enable OpenPGP encryption.
2. **Import Your GPG Keys**
    - Thunderbird automatically detects your existing GPG key pair.
    - If needed, import the key manually:

```
gpg --export -a "Your Name" > publickey.asc
```

- Load the key into Thunderbird.

3. **Exchange Public Keys**
   - Share your public key with the recipient.
   - Import their public key into Thunderbird or GPG:

```
gpg --import recipientkey.asc
```

   - This ensures both parties can encrypt messages for each other.
4. **Compose an Encrypted Email**
   - Click "Write" in Thunderbird.
   - Enter the recipient's email address.
   - Enable encryption by clicking the **Encrypt** icon (lock symbol).
   - Optionally enable a **Digital Signature** for authenticity.
5. **Send the Email**
   - Click "Send".
   - The message is encrypted automatically using the recipient's public key.
   - Only the recipient's private key can decrypt and read the email.
6. **Recipient Decrypts the Email**
   - The recipient receives ciphertext.
   - Their private key decrypts it automatically inside Thunderbird.

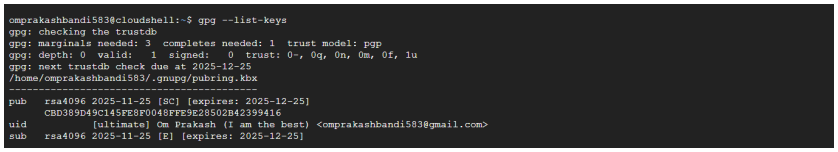

Figure 2: Encrypting an Email in Thunderbird

Privacy Tools from prism-break.org

For improving online privacy, security, and data protection, the following five tools were selected from `prism-break.org`. Each tool focuses on reducing tracking, surveillance, and unauthorized data access.

1. TOR Browser

TOR Browser is a privacy-focused web browser that routes internet traffic through the Tor network, making it difficult for websites, ISPs, or governments to trace the user's identity or location. It hides the user's IP address and prevents fingerprinting to a large extent. TOR also blocks trackers and cookies by default, ensuring anonymous browsing. It is especially useful for accessing restricted content and avoiding surveillance. However, due to multiple relays, browsing speed may be slower than normal browsers.

2. Signal

Signal is an open-source messaging application that provides end-to-end encryption for text messages, voice calls, video calls, and media sharing. Messages are encrypted in such a way that even Signal's servers cannot access their content. The app does not store user data, metadata, or contacts on its servers, ensuring strong privacy. Signal uses the Signal Protocol, one of the most secure encryption standards. It is free, cross-platform, and recommended by security experts like Edward Snowden.

3. KeePassXC

KeePassXC is an open-source password manager that securely stores passwords in an encrypted offline database. Instead of remembering multiple passwords, users can manage all credentials with a single master password. The database uses strong AES-256 encryption, making unauthorized access extremely difficult. KeePassXC supports password generation, autofill, and multi-device syncing through secure storage options. Since data is stored locally, there is no risk of cloud-based breaches.

4. DuckDuckGo

DuckDuckGo is a privacy-focused search engine that does not track users, store search history, or profile individuals for targeted ads. Unlike traditional search engines, it does not collect personal data or link queries to specific identities. It blocks trackers and ensures encrypted connections whenever possible through its browser extensions. DuckDuckGo provides unbiased search results without personalization filters. It is a strong alternative for users looking to minimize online surveillance.

5. ProtonMail

ProtonMail is an encrypted email service based in Switzerland, a country with strong privacy laws. It provides end-to-end encryption, meaning only the sender and recipient can read the email content. ProtonMail does not log IP addresses or store readable messages on its servers. It offers self-destructing emails, secure key management, and zero-access architecture. The service is open-source and accessible via web and mobile apps, making secure communication simple and user-friendly.

Open Source License Used

For this project, I selected the **MIT License**, which is one of the most widely used and permissive open-source licenses. It allows developers to freely use, modify, distribute, and integrate the software with minimal restrictions.

Why I Chose the MIT License

The decision to use the MIT License was based on the following reasons:

- **Simplicity:** The license is short, easy to understand, and legally clear.
- **Flexibility:** Allows anyone to reuse or modify the project for personal or commercial purposes.
- **Low Restrictions:** Only requires attribution to the original author, with no obligations to open-source modified versions.
- **Industry Standard:** Popular among developers and used by major projects like React, Node.js, and jQuery.
- **Encourages Collaboration:** Makes it easier for others to contribute without legal barriers.

Licensing Terms

The core terms of the MIT License include:

- Permission to use, copy, modify, merge, publish, distribute, and sublicense the software.
- Users must include the original copyright notice and license text in any copies or derivative works.
- The software is provided **"as is"** without any warranty or liability.
- The author is not responsible for damages, claims, or losses arising from the software's use.

These terms ensure that the software remains open and usable while protecting the developer from legal risks.

Project Screenshot

A screenshot of the project repository displaying the MIT License file is shown below.

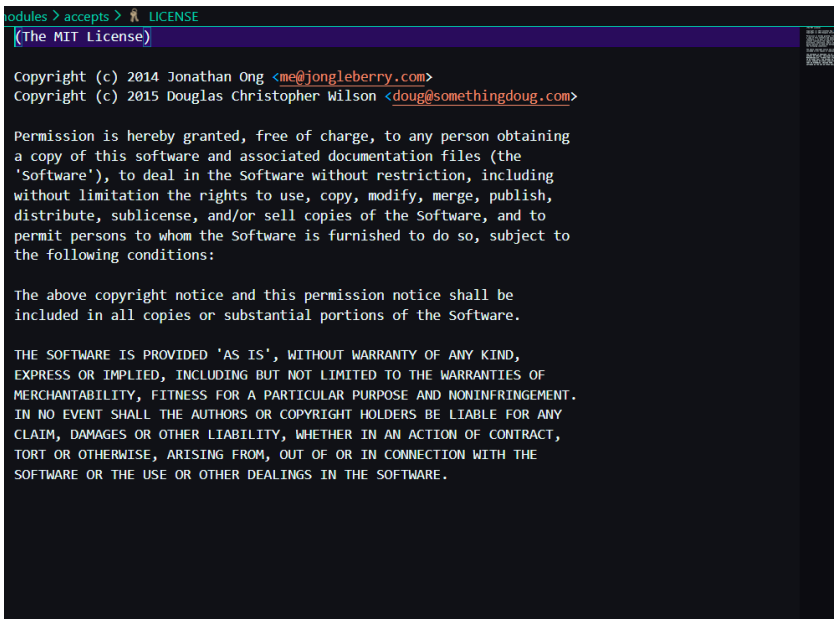

Figure 3: Project Repository Showing MIT License

Self-Hosted Server

About the Server

For this project, I self-hosted the **Manifest Server** locally on my own machine. A self-hosted server means the service runs on a personal system instead of relying on a third-party cloud provider. This provides more privacy, control, and ownership over data. The server allows hosting content, accessing files locally, and testing web-based services in a secure environment without exposing them to the public internet. Running the server locally also

helped me understand networking, hosting, configuration, and server management concepts.

Installation Steps

The following steps were used to install and launch the Manifest self-hosted server on Ubuntu:

1. **Updated System Packages**

   ```
   sudo apt update && sudo apt upgrade
   ```

2. **Installed Required Dependencies** (e.g., Git, Node.js, or Python depending on the server stack)

   ```
   sudo apt install git curl
   ```

3. **Cloned the Manifest Server Repository**

   ```
   git clone <repository-link>
   cd manifest-server
   ```

4. **Installed Project Dependencies**

   ```
   npm install
   ```

5. **Started the Local Server**

   ```
   npm start
   ```

6. **Accessed the Server Locally** Opened the browser and navigated to:

```
http://localhost:3000
```

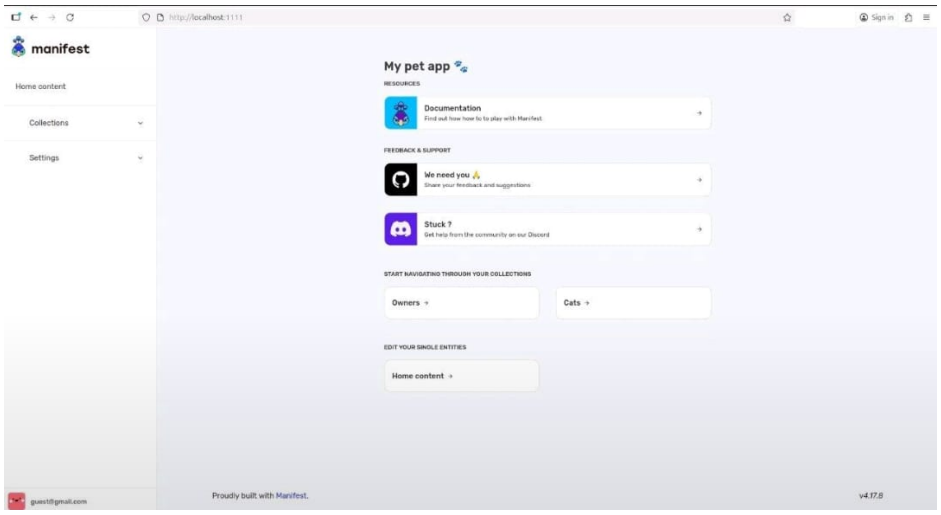

Figure 4: Local Host

Localized Document / Poster

A localized poster/document was created to explain the self-hosted server setup, its purpose, and usage in a simple and visually clear format. This poster highlights:

- What a self-hosted server is
- Why hosting locally improves privacy
- Steps to access the Manifest server
- Screenshots of the running interface

How You Used It

After successfully hosting the Manifest server locally, I used it for the following purposes:

- **Testing the Server Environment:** Verified that the service runs correctly without relying on cloud hosting.
- **Accessing Files/Services Locally:** Interacted with the Manifest interface through the browser.
- **Learning Server Management:** Practiced starting, stopping, and monitoring the server.
- **Privacy and Security:** Since the server was local, no external network had access to my data.
- **Experimentation:** Made configuration tweaks and observed changes in real-time.

This hands-on experience helped me understand how local hosting works and how applications can be deployed privately on personal hardware.

Open Source Contributions

During this project, I contributed to open-source development by identifying issues, fixing bugs, and submitting Pull Requests (PRs) to public repositories. These contributions helped me understand real-world collaboration, version control, and code review processes.

PR #1: Contribution no.1

- **Changes Done:** [first contribution].
- **PR Link:** `https://github.com/firstcontributions/first-contributions/pull/106319`

- **Status:** Merged

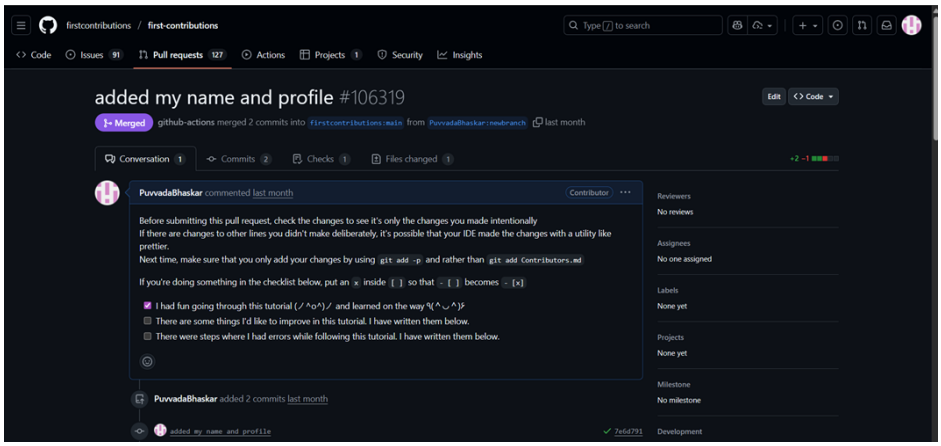

Figure 5: Pull Request #1

PR #2: Contributed a Java Program

- **Issue Summary:** solved a java program
- **What I Solved:** I have added a new java Program
- **PR Link:** https://github.com/DhanushNehru/Hacktoberfest2025/pull/971
- **Status:** Open / Under Review

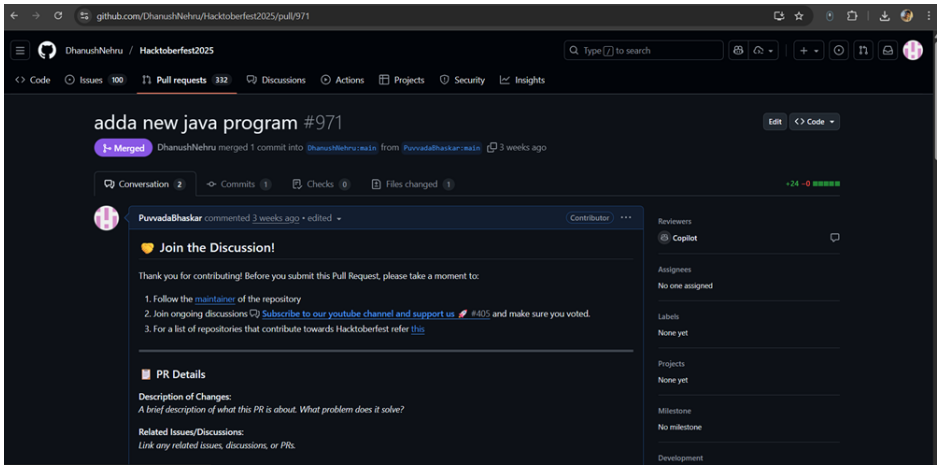

Figure 6: Enter Caption

Figure 7: Pull Request #2

PR #3: Code Optimization

- **Issue Summary:**Add question on undefined behavior with string literals
- **What I Solved:** added a new quiz which help the repo owner
- **PR Link:** `https://github.com/Ebazhanov/linkedin-skill-assessments-quizzes/pull/7229`
- **Status:** Merged

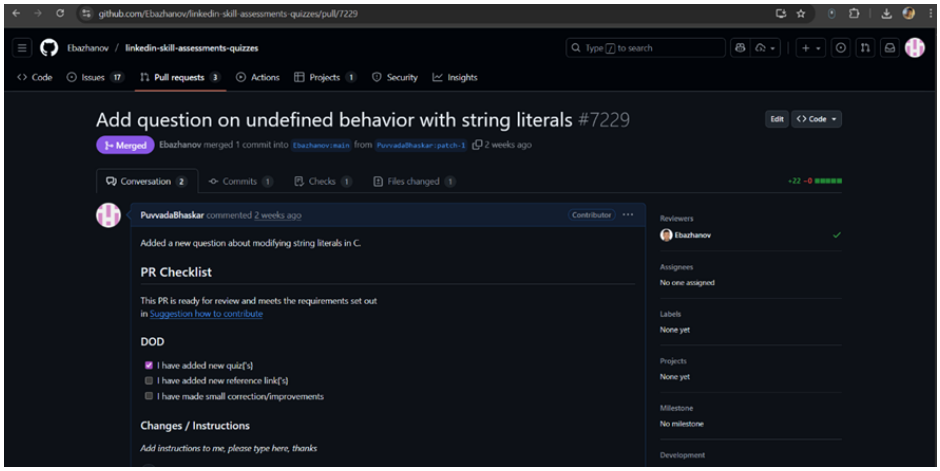

Figure 8: Enter Caption

Figure 9: Pull Request #3

LinkedIn Posts

Add 3 links:

1. Self Hosting Post
2. PR Merge Post
3. Blog Post