Chapter 3: Internship Activities

3.1. Roles and Responsibilities

While working as a Backend Python Developer intern for Khalti, my main focus was in the development and refinement of a various feature, which involved designing and implementing backend solutions using Python and Django, ensuring these systems were robust, scalable, and seamlessly integrated with the existing digital wallet platform.I had the following tasks:

i) **Code Development and Testing**:
I developed new backend features and enhanced existing functionalities using Python and Django, and I conducted thorough testing to ensure robustness and reliability.

ii) **System Maintenance and Optimization**:
I carried out routine maintenance and optimization tasks to improve the efficiency and scalability of the backend systems.

iii) **Database Management**:
I managed database schemas and integrated data storage solutions, ensuring data consistency and security.

iv) **Implementation of Security Protocols**:
I implemented and reviewed security measures to safeguard the digital wallet against potential threats and vulnerabilities.

v) **Collaboration with Other Teams**:
I maintained close collaboration with the front-end development team and other technical departments to ensure seamless integration and functionality across platforms.

vi) **Documentation of Development Processes**:
I created and maintained comprehensive documentation of all development processes, changes, and upgrades, providing a clear reference for future development efforts and troubleshooting.

3.2. Weekly log

Following table shows the weekly activities the intern performed throughout their internship period.

Table 3.1 : Weekly Log

Week	Activities
Week 1	• Introduced to the team and company culture. • Setup of personal development environment and necessary software. • Overview of the digital wallet codebase with a senior developer. • Attended workshops on Python and Django, the core technologies used.
Week 2	• Studied existing email functionalities within the digital wallet. • Reviewed documentation of related backend processes. • Participated in daily stand-ups to discuss ongoing projects. • Learned about security protocols relevant to user data handling.

Week 3	• Began coding the email update feature in the user profile section. • Implemented backend logic for sending verification emails. • Set up database migrations to handle new email data securely. • Developed error handling routines for failed email updates.
Week 4	• Began coding the email update feature in the user profile section. • Implemented backend logic for sending verification emails. • Set up database migrations to handle new email data securely. • Developed error handling routines for failed email updates.
Week 5	• Integrated the new email functionality with the frontend interface. • Conducted preliminary testing with dummy data to ensure stability. • Identified bugs and issues in the initial deployment. • Refined the API responses for better user feedback on errors.
Week 6	• Submitted code for peer review and incorporated feedback. • Optimized database queries for faster email updates. • Enhanced security measures based on latest best practices. • Wrote comprehensive unit tests for the new features.
Week 7	• Deployed the feature in a controlled test environment. • Monitored user interactions and collected feedback from test users. • Updated project documentation to include new features and changes. • Prepared a rollback plan for potential deployment issues.
Week 8	• Made final adjustments based on user feedback and testing results. • Conducted a final review with the project team and stakeholders. • Officially launched the email change feature to production. • Completed a detailed handover to the maintenance team, ensuring they are equipped to manage and troubleshoot the feature.

3.3. Description of the Project(s) Involved During Internship

One of the highlights which really helped me in understanding whole software development process was working on two minor projects during my internship both using django and django rest framework.

3.3.1. Allow users to securely edit/change email address

In this project, my main goal was to streamline the process of editing/verifying email address focusing on ease of use and security in mind.

i) The project is done using Django, Django Rest Framework, Django Templates, and Celery.
ii) The project involved creating a new API endpoint for users to change their email address.
iii) The project also involved creating a Celery task to send an email verification link to the new email address.

iv) The system was designed to ensure that the email change process was secure and user-friendly.

3.3.1.1 Functional Requirement

The functional requirements for a system describe what the system should do. Those requirements depend on the kind of software being developed and the expected software users. These are statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in specific situations.

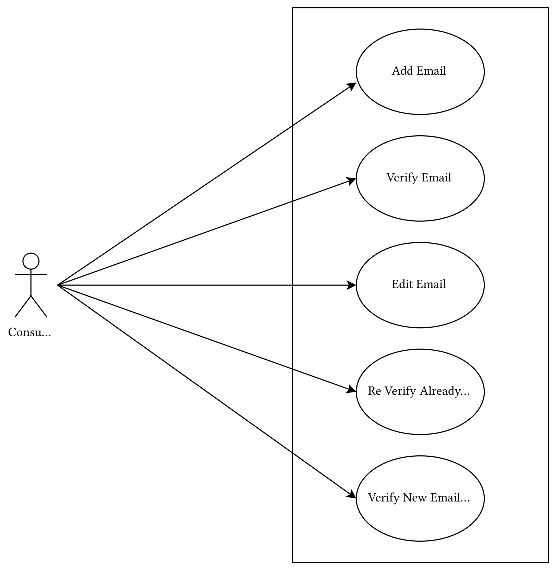


Figure 3.1 : Organizational Hierarchy

The Figure 3.1 represents a consumer and system interaction that shows the basic working features of the application itself, where the user can change their email address and verify it.

Use Case Description

Use Case ID	User Case Name	Actor	Description	Preconditions	Main Flow	Alternative Flow	Postconditions
UC01	View Profile	Consumer	Users can view additional options in their profile.	User is logged into the Khalti App and is in the Profile section.	i) User accesses profile.2. User views 3 dots for more options.	None	User sees 3 dots.
UC02	Add Email	Consumer	Allows the user to add an email address to their profile.	User is in the Profile section of the Khalti App.	i) User clicks on 3 dots.2. User selects “Add Email”.3. User enters	i) Invalid email format.2. Email linked to another account.	Email is added pending verification.

Use Case ID	User Case Name	Actor	Description	Preconditions	Main Flow	Alternative Flow	Postconditions
					email and submits for authentication.		
UC03	Authenticate Email Addition	Consumer	Handles authentication required for adding an email.	User has entered a valid email address.	i) User is prompted for authentication.2. User completes authentication.	i) Biometric fails, try password.2. Incorrect password input.	Email is authenticated, verification link sent.
UC04	Update Email	Consumer	Allows the user to change their email address.	User has previously added an email.	i) User accesses "Update Email" from profile.2. User enters new email and authenticates.	Same as Add Email use case.	Email update process initiated.
UC05	Handle Authentication Failures	Consumer	Manages repeated authentication failures.	User has failed authentication attempts.	User attempts to authenticate and fails.	User fails multiple times leading to lock-out.	User is temporarily locked out after specified failures.
UC06	Handle Expired Verification Link	Consumer	Manages scenarios where a user clicks an expired verification link.	User has received a verification link.	User clicks expired verification link.	None	User is directed to a webpage to request a new link.
UC07	Resend Verification Email	Consumer	Allows users to resend the verification email if not received.	User has not received the initial verification email.	User selects the option to resend verification email.	None	Verification email is resent.

Use Case ID	User Case Name	Actor	Description	Preconditions	Main Flow	Alternative Flow	Postconditions
UC08	Handle Network Issues During Email Submission	Consumer	Manages scenarios with network connectivity issues during email submission.	User is adding an email.	User faces network issues during the submission.	None	Error message is displayed, advises checking network.

3.3.1.2 Non-Functional Requirement
Non-functional requirements are requirements that are not directly concerned with the specified function delivered by the system.

- **Maintainability**:
  Adhere to coding standards that promote readability and maintainability.
- **Compliance**:
  Ensure the system complies with relevant data protection regulations (like GDPR, if applicable) regarding user data.

3.3.1.3 Feasibility Study
Before starting the project, a feasibility study is carried out to measure the system's viability. A feasibility study is necessary to determine if creating a new or improved system is friendly with the cost, benefits, operation, technology, and time. The following are the feasibility concerns in this project:

i) **Technical Feasibility**:

The technical feasibility of enhancing the email edit/update feature in a Django-based system considers both the current technological framework and specific security requirements essential in the financial sector. Given that Django is robust in managing secure user interactions and database modifications (necessary for updating email addresses), the existing infrastructure is likely sufficient. However, due to the sensitive nature of financial data, the security implementations need to be rigorously evaluated. This includes ensuring encryption for data transmissions and safeguarding against vulnerabilities specific to financial applications, such as phishing and session hijacking.

i) **Operational Feasibility**:
From an operational feasibility standpoint, integrating this feature into Khalti's digital wallet platform is seamless and minimally disruptive to existing operations. Since digital wallets require high reliability and user trust, any changes that involve user account details, like email addresses, must be handled with extreme caution to avoid eroding trust or introducing errors. The introduction of the feature should be accompanied by comprehensive user documentation and possibly a brief tutorial within the app to facilitate adoption. Additionally, since this feature is fundamental to account security, ensure that your customer support team is well-prepared to address any issues that arise quickly and effectively. This might involve specialized training and updating internal operational protocols.

ii) **Economic Feasibility**:
The economic feasibility for Khalti involves a detailed analysis of costs versus anticipated

benefits. The direct costs include development, testing, additional security measures, and training. Operationally, the feature should lead to a reduction in support costs, as users can self-manage their email details, potentially lowering the volume of support requests related to account access issues. From a benefits perspective, enhancing user autonomy and security can improve customer satisfaction and retention—a crucial metric in the competitive fintech space. Calculating the return on investment should factor in these indirect benefits, such as enhanced user trust and reduced risk of security breaches, which can have significant financial implications. As a fintech entity, projecting the feature's impact on enhancing regulatory compliance and reducing fraud incidents could further justify the investment.

3.3.1.4 System Design

System design defines the components, modules, interfaces, and data for a system to satisfy specified requirements. It can also be defined as creating or altering systems and the processes, practices, models, and methodologies used to develop them. The main objective of the detailed system design is to prepare a system blueprint that meets the goals of the conceptual system design requirements. The system designs for building this project include database schema, input-output design, class diagram, sequence diagram, and activity diagram.

3.3.1.4 Architectural Design

The architectural design shows the architecture of the overall system. The system is based on MVC architecture.

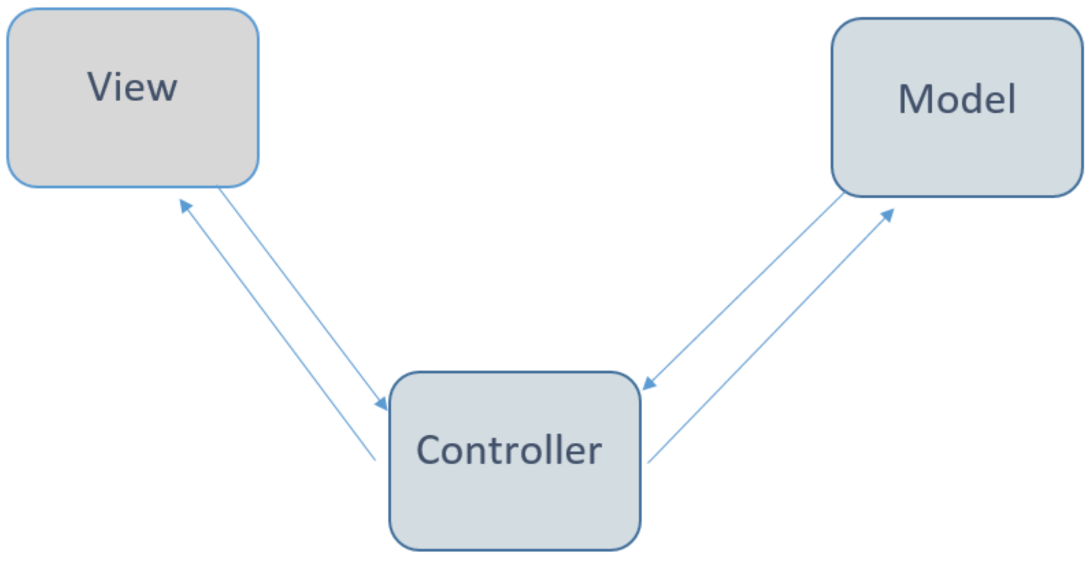


Figure 3.2 : Architectural Design of the System

3.3.1.5 Database Design

The following database schema is the general structure used in the application.

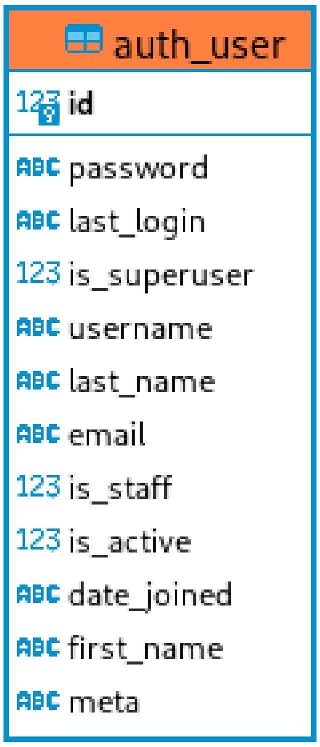


Figure 3.3 : Table of User Model

3.3.1.6 Class Diagram

The class diagram describes the relationships and source code dependencies among other classes.

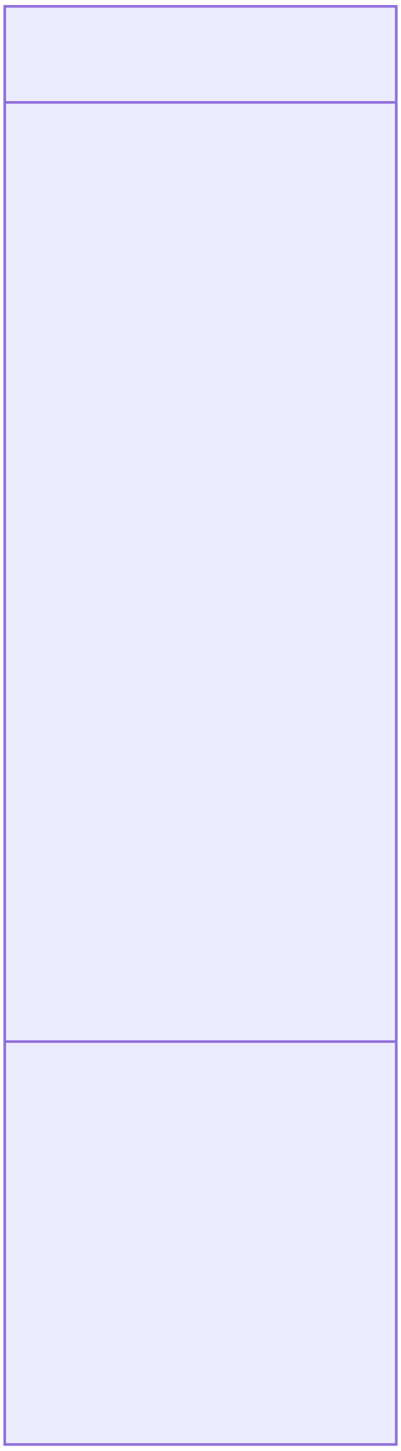


Figure 3.3 : Class Diagram of User Model

The Figure 3.3 illustrates the class diagram of the system. There is only one class, User, which is responsible for handling user data and operations on itself.

3.3.1.7 Sequence Diagram

A sequence diagram is an interaction diagram because it describes how and in what order a group of objects works together. Sequence diagrams are sometimes known as event diagrams. The sequence diagram of the system is shown below.

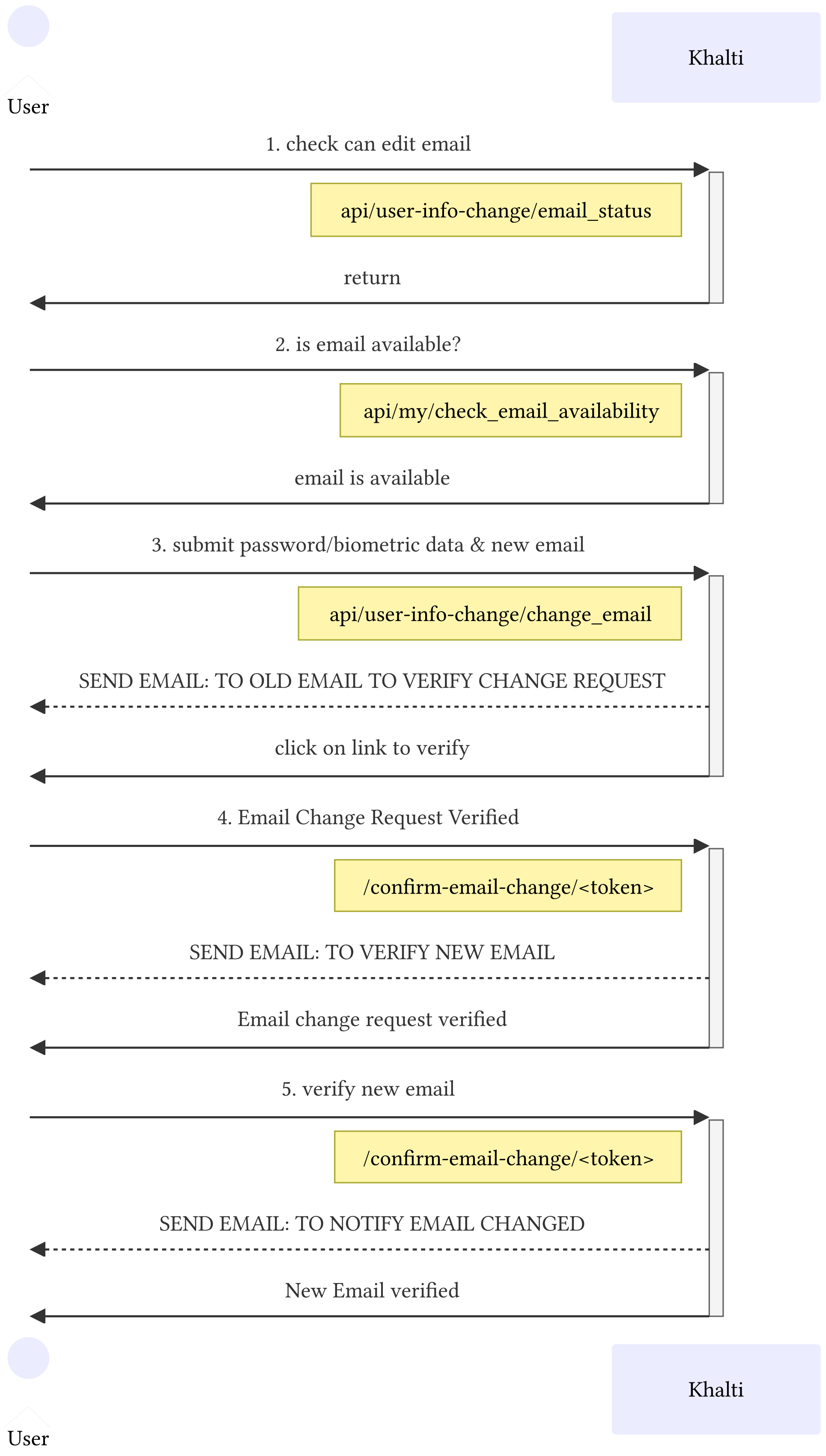

User
Khalti
1. check can edit email
api/user-info-change/email_status
return
2. is email available?
api/my/check_email_availability
email is available
3. submit password/biometric data & new email
api/user-info-change/change_email
SEND EMAIL: TO OLD EMAIL TO VERIFY CHANGE REQUEST
click on link to verify
4. Email Change Request Verified
/confirm-email-change/<token>
SEND EMAIL: TO VERIFY NEW EMAIL
Email change request verified
5. verify new email
/confirm-email-change/<token>
SEND EMAIL: TO NOTIFY EMAIL CHANGED
New Email verified
User
Khalti

Figure 3.3 : Class Diagram of User Model

The Figure 3.3 illustrates how the frontend system will interact with the backend api to change the email address of the user.

3.4. Tasks / Activities Performed

i) Pushing docker image to self hosted container registry

In this task, Docker images have been pushed to self hosted container registry using self hosted jenkins by running shell script. Given steps were being followed to do the task:

- Install Jenkins and it's dependencies using following shell script:

File: Jenkins-install.sh

```
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat/jenkins.io-2023.key
sudo dnf upgrade
# Add required dependencies for the jenkins package
sudo dnf install fontconfig java-17-openjdk
sudo dnf install jenkins
sudo systemctl enable jenkins
sudo systemctl start jenkins

JENKINS_PORT=8080
PERM="--permanent"
SERV="$PERM --service=jenkins"

firewall-cmd $PERM --new-service=jenkins
firewall-cmd $SERV --set-short="Jenkins ports"
firewall-cmd $SERV --set-description="Jenkins port exceptions"
firewall-cmd $SERV --add-port=$JENKINS_PORT/tcp
firewall-cmd $PERM --add-service=jenkins
firewall-cmd --zone=public --add-service=http --permanent
firewall-cmd --reload
```

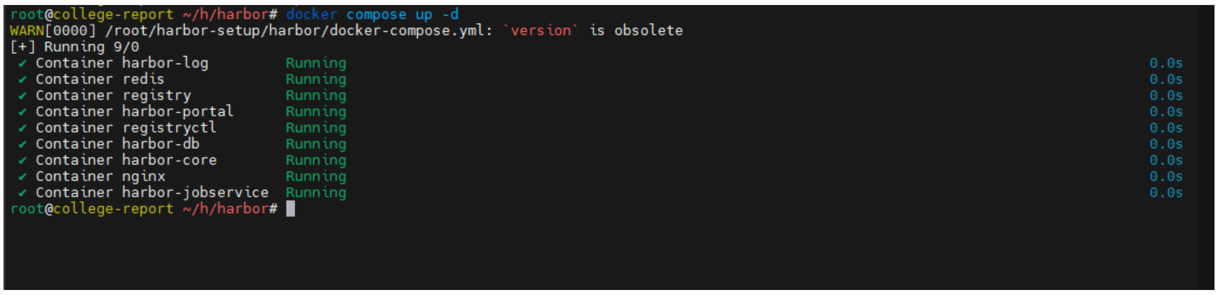
- Install and configure Harbor which is self hosted registry using docker with following script and create a new project from Harbor Dashboard.

File: Harbor-install-and-configure.sh

```
mkdir -p harbor-setup; cd harbor-setup
wget https://github.com/goharbor/harbor/releases/download/v2.11.0/harbor-online-
installer-v2.11.0.tgz
tar xvf harbor-online-installer-v2.11.0.tgz
openssl genrsa -out ca.key 4096

echo -e "\n------------\n Creating Root CA Certificate \n----------------\n "
openssl req -x509 -new -nodes -sha512 -days 3650 \
 -subj "/C=NP/ST=Bagmati Pradesh/L=Kathmandu/O=MeroOrganization/OU=Personal/
CN=merodomain.com" \
 -key ca.key \
 -out ca.crt

echo -e "\n------------\n Generating Server Certificate \n----------------\n"
openssl genrsa -out meroharbor.com.key 4096
openssl req -sha512 -new \
    -subj "/C=NP/ST=Mero State/L=Mero Thau/O=Mero Organization/OU=Personal/
CN=meroharbor.com" \
    -key meroharbor.com.key \
    -out meroharbor.com.csr

cat > meroharbor.com.v3.ext <<-EOF
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
[alt_names]
DNS.1=meroharbor.com
EOF

echo -e "\n------------\n Signing Server Certificate with CA's Certificate
\n----------------\n"
```

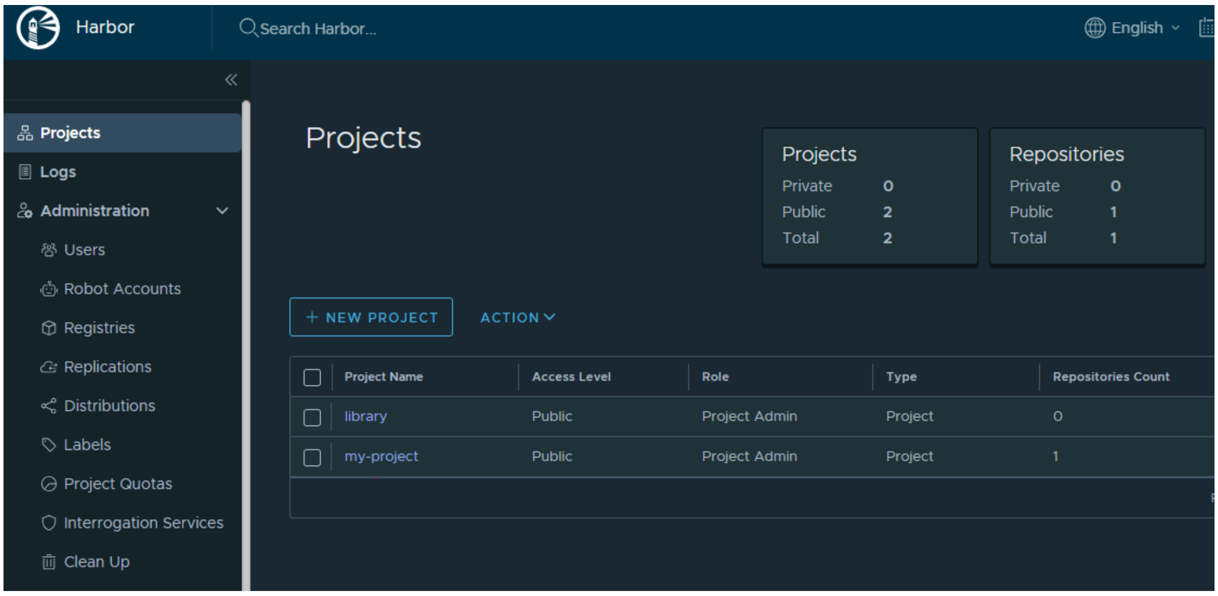
```
openssl x509 -req -sha512 -days 3650 \
    -extfile meroharbor.com.v3.ext \
    -CA ca.crt -CAkey ca.key -CAcreateserial \
    -in meroharbor.com.csr \
    -out meroharbor.com.crt

echo -e "\n------------\n Converting crt to cert \n----------------\n"
openssl x509 -inform PEM -in meroharbor.com.crt -out meroharbor.com.cert
mkdir -p /etc/docker/certs.d/meroharbor.com:443/
cp meroharbor.com.cert /etc/docker/certs.d/meroharbor.com:443/
cp meroharbor.com.key /etc/docker/certs.d/meroharbor.com:443/
cp ca.crt /etc/docker/certs.d/meroharbor.com:443/
cd harbor; cp harbor.yml.tmpl harbor.yml
sed -i 's|hostname: reg.mydomain.com|hostname: meroharbor.com|; s|certificate: /your/
certificate/path|certificate: /etc/docker/certs.d/meroharbor.com:443/
meroharbor.com.cert|; s|private_key: /your/private/key/path|private_key: /etc/docker/
certs.d/meroharbor.com:443/meroharbor.com.key|' harbor.yml
./prepare
docker compose up -d
```

Following picture provide the output of the above shell script:



```
root@college-report ~/h/harbor# docker compose up -d
WARN[0000] /root/harbor-setup/harbor/docker-compose.yml: `version` is obsolete
[+] Running 9/0
 ✓ Container harbor-log         Running                                                                     0.0s
 ✓ Container redis              Running                                                                     0.0s
 ✓ Container registry           Running                                                                     0.0s
 ✓ Container harbor-portal      Running                                                                     0.0s
 ✓ Container registryctl        Running                                                                     0.0s
 ✓ Container harbor-db          Running                                                                     0.0s
 ✓ Container harbor-core        Running                                                                     0.0s
 ✓ Container nginx              Running                                                                     0.0s
 ✓ Container harbor-jobservice  Running                                                                     0.0s
root@college-report ~/h/harbor#
```

Figure 3.1 : Harbor Installation



Projects	
Private	0
Public	2
Total	2

Repositories	
Private	0
Public	1
Total	1

Project Name	Access Level	Role	Type	Repositories Count
library	Public	Project Admin	Project	0
my-project	Public	Project Admin	Project	1

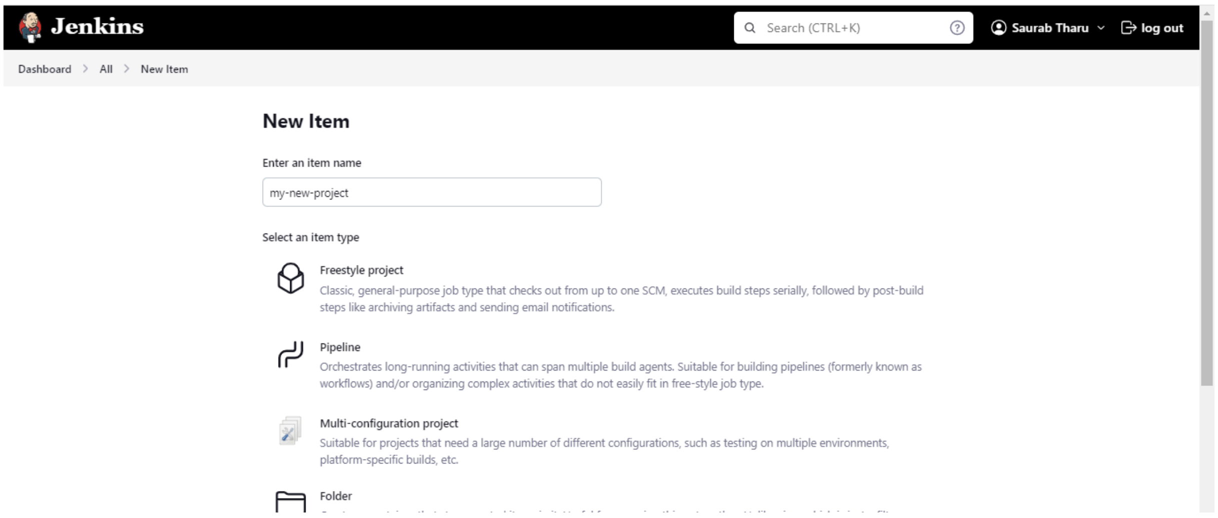
Figure 3.2 : Harbor Project Creation

- Create Dockerfile to build docker image

File: Dockefile

```
FROM nginx:latest
COPY ./mysite /usr/share/nginx/html/
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Step 4: Create a Freestyle Project in Jenkins


Jenkins

Dashboard > All > New Item

New Item

Enter an item name

my-new-project

Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

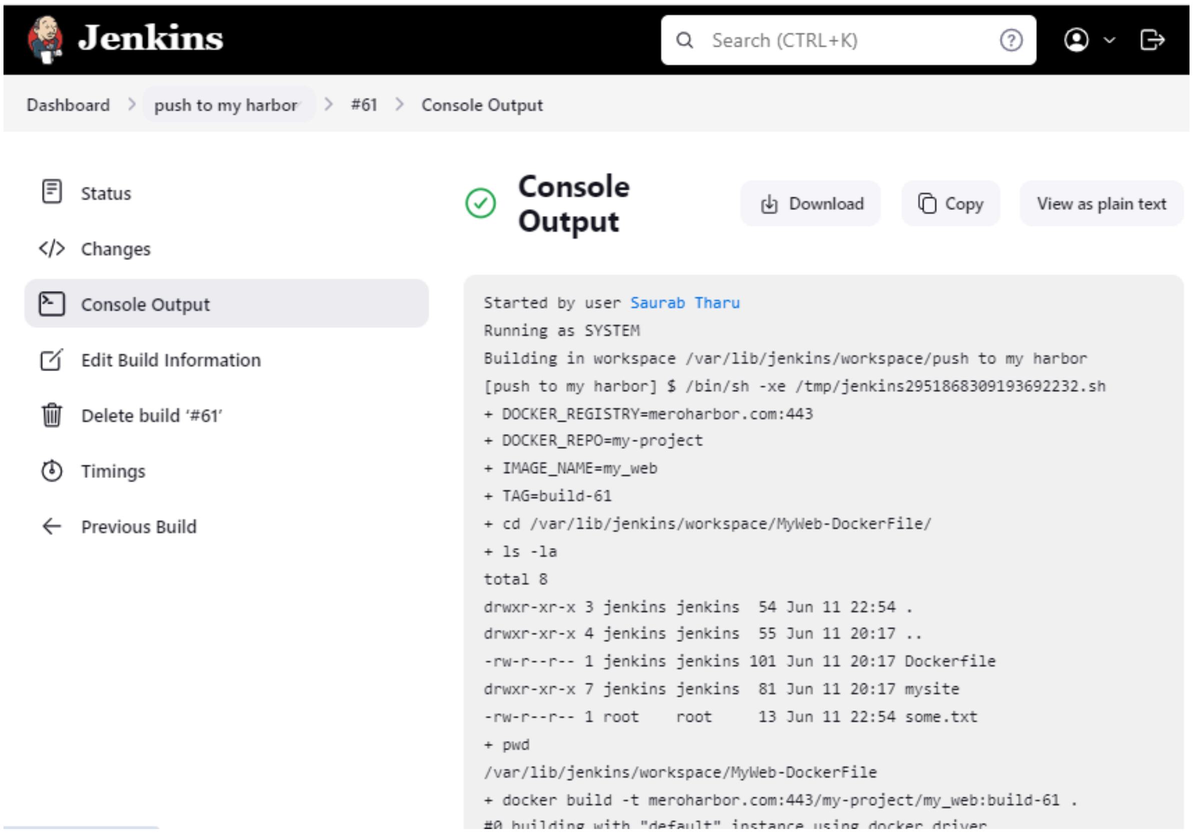
Folder

Figure 3.3 : Creating Jenkins Freestyle Project

- Configure the jenkins project by adding **"Exectue shell"** build step and use the following shell script and save the configuration.

```
DOCKER_REGISTRY='meroharbor.com:443'
DOCKER_REPO='my-project'
IMAGE_NAME='my_web'
TAG="build-${BUILD_NUMBER}"

cd /var/lib/jenkins/workspace/MyWeb-DockerFile/
ls -al

docker build -t ${DOCKER_REGISTRY}/${DOCKER_REPO}/${IMAGE_NAME}:${TAG} .

# here password to login is stored in some.txt file
cat some.txt | docker login ${DOCKER_REGISTRY} -u saurab --password-stdin

docker push ${DOCKER_REGISTRY}/${DOCKER_REPO}/${IMAGE_NAME}:${TAG}
docker logout
```

- Finally build the project and check the **"Console Output"** to verify if build was successful or not.



```
Started by user Saurab Tharu
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/push to my harbor
[push to my harbor] $ /bin/sh -xe /tmp/jenkins2951868309193692232.sh
+ DOCKER_REGISTRY=meroharbor.com:443
+ DOCKER_REPO=my-project
+ IMAGE_NAME=my_web
+ TAG=build-61
+ cd /var/lib/jenkins/workspace/MyWeb-DockerFile/
+ ls -la
total 8
drwxr-xr-x 3 jenkins jenkins  54 Jun 11 22:54 .
drwxr-xr-x 4 jenkins jenkins  55 Jun 11 20:17 ..
-rw-r--r-- 1 jenkins jenkins 101 Jun 11 20:17 Dockerfile
drwxr-xr-x 7 jenkins jenkins  81 Jun 11 20:17 mysite
-rw-r--r-- 1 root    root     13 Jun 11 22:54 some.txt
+ pwd
/var/lib/jenkins/workspace/MyWeb-DockerFile
+ docker build -t meroharbor.com:443/my-project/my_web:build-61 .
#0 building with "default" instance using docker driver
```

Figure 3.4 : Jenkins Console Output

ii) Dockerizing nginx hosted website and providing SSL certificate to the website

Hosting a website using Nginx and securing it with an SSL certificate is one of the fundamental task DevOps needs to understand. This involved preparing a Dockerfile and configuring the Nginx server to serve the website content and setting up virtual hosts. To ensure secure communication, SSL certificates were generated and installed. openssl tool is used for generating certificates which includes certificate.crt file and private key .key file. Following files were used for dockerization of nginx website:

File: generate_certificate.sh

```
#!/bin/bash

set -x
openssl genrsa -out ca.key 4096

echo -e "\n------------\n Creating Root CA Certificate \n----------------\n "
openssl req -x509 -new -nodes -sha512 -days 3650 \
 -subj "/C=NP/ST=Bagmati Pradesh/L=Kathmandu/O=MeroOrganization/OU=Personal/
CN=meroCA.com" \
 -key ca.key \
 -out ca.crt

echo -e "\n------------\n Generating Server Certificate \n----------------\n"
openssl genrsa -out merowebsite.com.key 4096
openssl req -sha512 -new \
    -subj "/C=NP/ST=Mero State/L=Mero Thau/O=Mero Organization/OU=Personal/
CN=merowebsite.com" \
    -key merowebsite.com.key \
    -out merowebsite.com.csr

cat > merowebsite.com.v3.ext <<-EOF
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

[alt_names]
DNS.1=merowebsite.com
EOF
```

```
echo -e "\n------------\n Signing Server Certificate with CA's Certificate
\n----------------\n"
openssl x509 -req -sha512 -days 3650 \
    -extfile merowebsite.com.v3.ext \
    -CA ca.crt -CAkey ca.key -CAcreateserial \
    -in merowebsite.com.csr \
    -out merowebsite.com.crt
```

File: nginx.conf

```
server {
    listen       443;
    server_name  merowebsite.com;

    ssl on;
    ssl_certificate     /usr/share/nginx/certificates/merowebsite.com.crt;
    ssl_certificate_key /usr/share/nginx/certificates/merowebsite.com.key;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }
}
```

File: Dockerfile

```
FROM nginx:1.10.1-alpine
COPY jd /usr/share/nginx/html
COPY merowebsite.com.crt /usr/share/nginx/certificates/
COPY merowebsite.com.key /usr/share/nginx/certificates/
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 443
CMD ["nginx", "-g", "daemon off;"]
```

iii) Configuring up load balancing

Suppose we have four website hosted using nginx at port 1111, 2222, 3333, 4444 with following nginx configuration

File: website-for-load-balancing.conf

```
############################################################
#   This is for loadbalancing
############################################################

# localhost:1111
server {
    listen       1111;
    server_name  localhost;
    root         /usr/share/nginx/digital-trend;
    index  index.html index.htm
}

# localhost:2222
server {
    listen       2222;
    server_name  localhost;
    root         /usr/share/nginx/jackpiro;
    index  index.html index.htm
}

# localhost:3333
server {
    listen       3333;
    server_name  localhost;
    root         /usr/share/nginx/memorial;
    index  index.html index.htm
}

# localhost:4444
server {
    listen       4444;
    server_name  localhost;
    root         /usr/share/nginx/shiphile;
    index  index.html index.htm
}
```

a) Using Nginx

The following Nginx configuration can be used to set up a load balancer that distributes traffic across multiple backend servers. This setup ensures that requests are balanced between the servers specified in the upstream block.

File: haproxy.cfg

```
upstream myapp1 {
    server localhost:1111;
    server localhost:2222;
    server localhost:3333;
    server localhost:4444;
}
server {
    listen 80;
    server_name nginx-load-balancer.com;              # domain-name
    location / {
        proxy_pass http://myapp1;
    }
}
```

b) Using HAProxy

The following configuration can be used for HAProxy to load balance traffic across multiple nodes, ensuring that if any node goes down, the other nodes will continue to serve the website. This setup distributes incoming traffic using the round-robin algorithm and checks the health of each node to maintain high availability.

File: haproxy.cfg

```
frontend main
    bind *:80
    timeout client 60s
    mode http
    default_backend backend_name # <- name of backend specified below
 backend backend_name
    timeout connect 30s
    timeout server 100s
    mode http
    balance roundrobin
    http-request set-header Host load-balanced-website.com
    server node1  127.0.0.1:1111  check
    server node2  127.0.0.1:2222  check
    server node3  127.0.0.1:3333  check
    server node4  127.0.0.1:4444  check
```

v) Configuring k3s for synchronizing deployed website with code in Gitea

K3s, a lightweight Kubernetes distribution, was used for managing containerized applications efficiently. Synchronizing a deployed website with a GitHub repository ensures continuous integration and deployment, allowing the website to reflect the latest code changes automatically.At first setting up the environment involves installing k3s, a lightweight Kubernetes distribution that simplifies cluster setup and management. By running a single command, we install k3s and initiate the server. Configuring `kubectl` ensures that we can manage the k3s cluster efficiently, setting the stage for deploying applications. This streamlined setup process demonstrates k3s's ease of use and quick deployment capabilities.

To install k3s, execute the following command in your terminal. This command downloads and installs the k3s binary and starts the k3s server.

```
$ curl -sfL https://get.k3s.io | sh -
```

Deploying a sample website on k3s involves creating a deployment configuration file (`deployment.yaml`) that defines the desired state of the application, including the number of replicas, container images, and service specifications. Applying this configuration with `kubectl` deploys the website on the k3s cluster.

File: nginx-deploymentgit.yaml

```
apiapiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy-resturan
  namespace: nginx
  labels:
    app: nginx-cc
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-cc
  template:
    metadata:
      labels:
        app: nginx-cc
    spec:
      containers:
      - name: git-sync
        image: registry.k8s.io/git-sync/git-sync:v4.2.3
        volumeMounts:
        - name: www-data
          mountPath: /data
        env:
        - name: GITSYNC_REPO
          value: "http://192.168.33.11:3000/saurab/website-for-git-sync.git"
        - name: GIT_SYNC_BRANCH
          value: "main"
        - name: GITSYNC_ROOT
          value: /data
        - name: GIT_SYNC_DEST
          value: "html"
        - name: GITSYNC_ONE_TIME
          value: "false"
        - name: GIT_SYNC_COMMAND_AFTER
          value: "nginx -s reload"
        securityContext:
          runAsUser: 0
      - name: nginx-resturan
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: "/usr/share/nginx/"
          name: www-data
      volumes:
      - name: www-data
        emptyDir: {}
```

This `nginx-ingress.yaml` file defines an Ingress resource named nginx-ingress using the Traefik controller, with TLS termination via tls-secret for roltu.com. It routes all HTTP requests to roltu.com to the nginx-service on port 80.

File: ingnx-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    kubernetes.io/ingress.class: "traefik"
```

```
spec:
  tls:
    - secretName: tls-secret
      hosts:
        - "roltu.com"
  rules:
    - host: roltu.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: nginx-service
                port:
                  number: 80
```

This `nginx-service.yaml` defines a Kubernetes `Service` named `nginx-service` in the `nginx` namespace. It uses a `NodePort` to expose port 80, forwarding traffic from the service on port 80 to the pods labeled `app: nginx-cc` on port 80, accessible via node IP on port 30002.

File: nginx-namespace.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
    name: nginx
```

File: nginx-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: nginx
spec:
  type: NodePort
  selector:
    app: nginx-cc
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30002
```

Apply the configuration to the k3s cluster using kubectl using commands as below:

```
$ kubectl apply -f nginx-namespace.yaml
$ kubectl apply -f nginx-deploymentgit.yaml
$ kubectl apply -f nginx-ingress.yaml
$ kubectl apply -f nginx-service.yaml
```

The above deployment file to work we need a git repo so for that self hosted Gitea was used. To install the Gitea following bash script is used:

File: gitea-installation.sh

```
  #!/bin/bash
# Create a directory and navigate into it
mkdir gitea
cd gitea
wget -O gitea https://dl.gitea.com/gitea/1.22.0/gitea-1.22.0-linux-amd64
chmod +x gitea
useradd git
groupadd gitservice
chown -R root:gitservice gitea
chmod g+s gitea
mkdir -p /usr/local/bin/gitea
mv gitea /usr/local/bin/gitea

cat > /etc/systemd/system/gitea.service <<-EOF
[Unit]
Description=Gitea (Git with a cup of tea)
After=syslog.target
After=network.target
#After=mysqld.service
#After=postgresql.service
#After=memcached.service
#After=redis.service

[Service]
# Modify these two values and uncomment them if you have
# repos with lots of files and get an HTTP error 500 because
# of that
#LimitMEMLOCK=infinity
#LimitNOFILE=65535
RestartSec=2s
Type=simple
```

```
User=git
Group=gitservice
WorkingDirectory=/usr/local/bin/gitea
ExecStart=/usr/local/bin/gitea/gitea web
Restart=always
Environment=USER=git HOME=/home/git
# If you want to bind Gitea to a port below 1024 uncomment
# the two values below
#CapabilityBoundingSet=CAP_NET_BIND_SERVICE
#AmbientCapabilities=CAP_NET_BIND_SERVICE

[Install]
WantedBy=multi-user.target
EOF

systemctl start gitea.service
systemctl enable gitea.service
```

After installation, the gitea can be accessed using ipaddress along with the port as below:

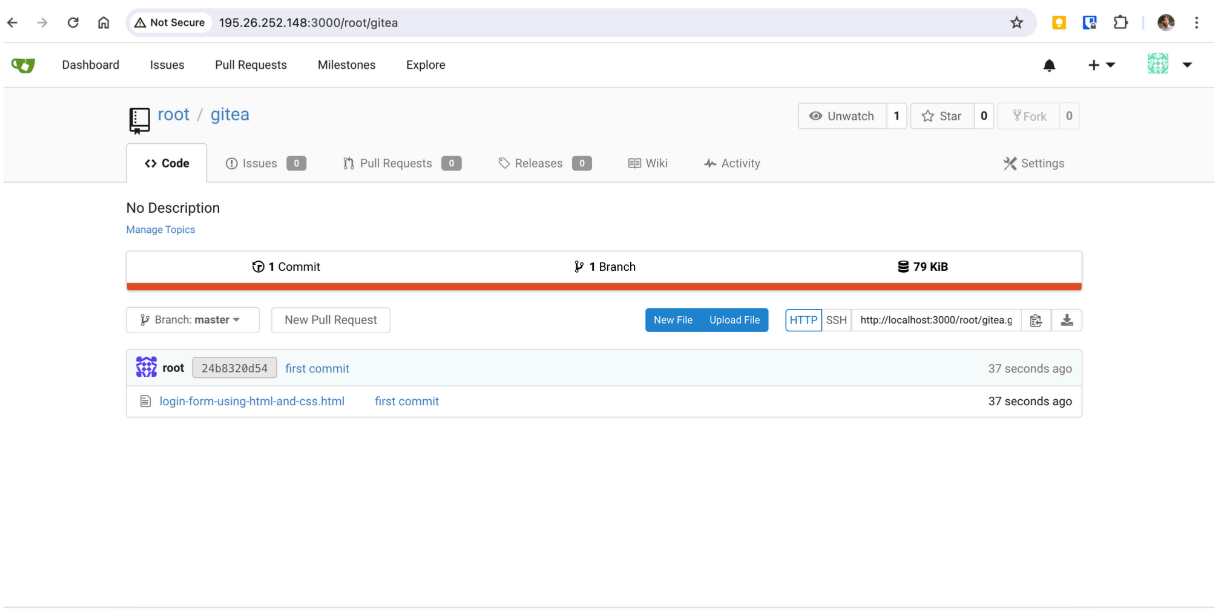


Figure 3.5 : Git Repo Locally Hosted