**EXPERIMENT-01**

**AIM: Write a code for simple User Registration form for an event.**

<!-- HTML CODE WITH INLINE CSS FOR REGISTRATION FORM -->

# Objective

To develop a basic user registration form using HTML and JavaScript, allowing users to submit their details for event registration.

# Software Requirements

* Code editor (VS Code or any)
* Web browser (Chrome, Firefox, etc.)

# Procedure

1. Create HTML File for User Registration Form

* Create a file named registration.html.
* Add the following code:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>HTML Registration Form</title>

<script src="script.js" defer></script>

<style> body {

font-family: Arial, sans-serif; margin: 0; padding: 0; display: flex; justify-content: center; align-items: center;

height: 100vh;

}

.main { background-color: #fff; border-radius: 15px;

box-shadow: 0 0 20px rgba(0, 0, 0, 0.2); padding:20px; width: 300px;

}

.main h2 {

color: #4caf50;

margin-bottom: 20px;

} label { display: block; margin-bottom: 5px; color: #555; font-weight: bold;

}

input[type="text"], input[type="email"], input[type="password"], select { width: 100%; margin-bottom: 15px; padding: 10px; box-sizing: border-box; border: 1px solid #ddd;

border-radius: 5px;

} button { padding: 15px; border-radius: 10px; border: none; background-color: #4caf50;

color: white; cursor: pointer; width: 100%;

font-size: 16px;

}

</style>

</head>

<body>

<div class="main">

<h2>Registration Form</h2>

<form action="" method="">

<label for="firstName">First Name</label>

<input type="text" id="first" name="first" required />

<label for="lastName">Last Name</label>

<input type="text" id="last" name="last" required />

<label for="email">Email</label>

<input type="email" id="email" name="email" required />

<label for="password">Password</label>

<input type="password" id="password" name="password" pattern="^(?=.\*\d)(?=.\*[a-zA-Z])(?=.\*[^a-zA-Z0-9])\S{8,}$" title="Password must contain at least one number, one alphabet, one symbol, and be at

least 8 characters long" required />

<label for="repassword">Re-type Password</label>

<input type="password" id="repassword" name="repassword" required />

<label for="mobile">Contact</label>

<input type="text" id="mobile" name="mobile" maxlength="10" required />

<label for="gender">Gender</label>

<select id="gender" name="gender" required>

<option value="male">

Male

</option>

<option value="female">

Female

</option>

<option value="other">

Other

</option> </select>

<button type="button" onclick="submitForm(); return false;">Register</button> <p id="confirmation"></p>

</form>

</div>

</body>

</html>

# Create JavaScript File for Form Handling

* Create a file named script.js.
* Add the following code:

function submitForm() {

var firstName = document.getElementById("first").value; var lastName = document.getElementById("last").value; var email = document.getElementById("email").value; var mobile = document.getElementById("mobile").value; var gender = document.getElementById("gender").value;

var password = document.getElementById("password").value; var repassword = document.getElementById("repassword").value;

if (!firstName || !lastName || !email || !password || !repassword || !mobile || !gender) { alert("Please fill all the fields."); return;

}

// Password match check if (password !== repassword) {

alert("Passwords do not match!"); return;

}

document.getElementById('confirmation').innerText =

'Thank you, ' + firstName + '! You have registered successfully.';

console.log('Registration Details:', { Name: firstName, Email: email, Phone: mobile

});

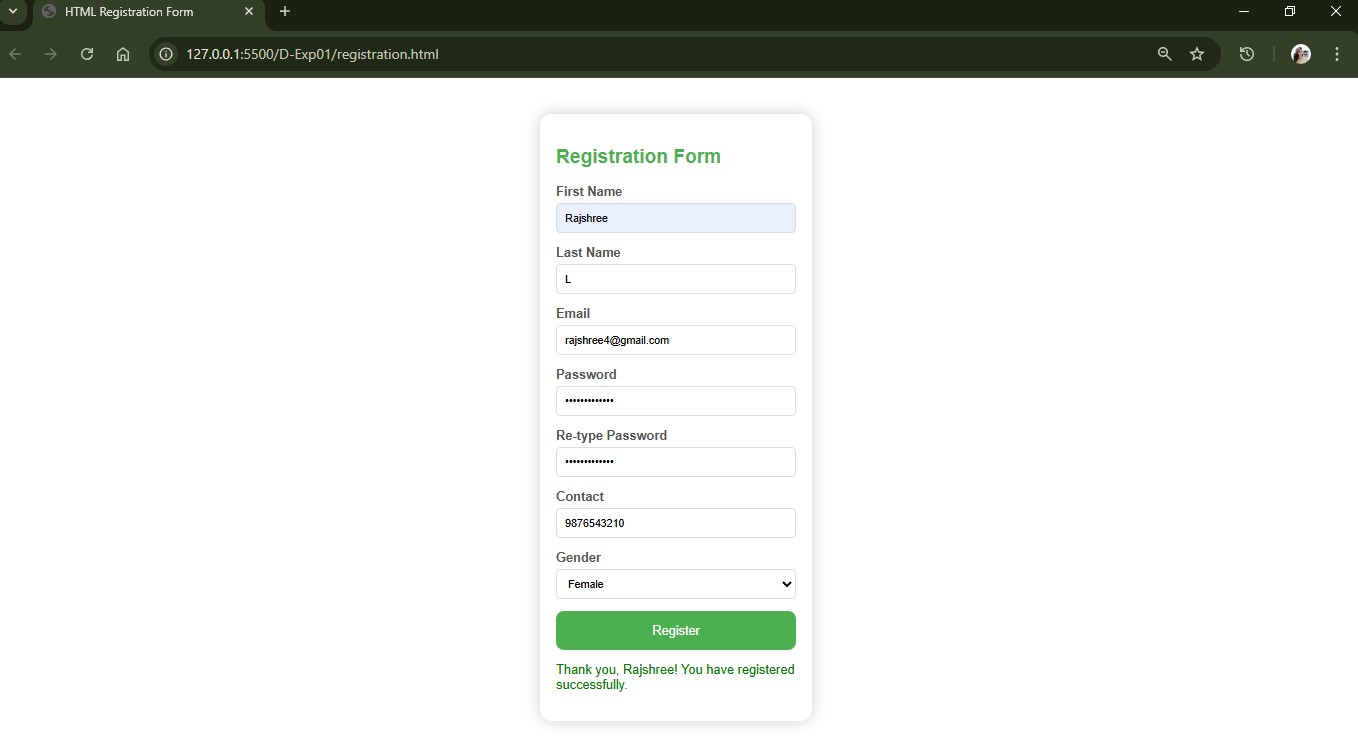
document.getElementById("confirmation").style.color = "green";

}

# Run and Test

* Open the registration.html file in a web browser.
* Fill in user details and click **Register**.
* Verify registration confirmation message is displayed.

# Output



**EXPERIMENT-02**

# AIM: Explore Git and GitHub Commands

# Objective

To explore and practice basic Git and GitHub commands for version control in a collaborative environment.

## Software Requirements

* Git installed ([https://git-scm.com/downloads)](https://git-scm.com/downloads)
* GitHub account ([https://github.com/)](https://github.com/)
* Code editor (VS Code or any)

## VERSION CONTROL SYSTEM

A **Version Control System (VCS)** is a tool that helps manage changes to files, particularly source code, over time. It tracks modifications, allows multiple people to collaborate on the same project, and ensures that a complete history of changes is maintained. VCS is essential for software development, but it can also be used for other types of documents or files.

### GIT

**Git** is a distributed version control system (VCS) designed to manage and track changes in source code during software development. Created by Linus Torvalds in 2005, Git is widely used by developers to collaborate on projects, keep track of changes, and manage multiple versions of their codebase.

### GITHUB

**GitHub** is a web-based platform and service that provides hosting for software development and version control using Git. It offers a collaborative environment where developers can manage and share their projects, track issues, and work together on code. GitHub is widely used in the software development community for both open-source and private projects.

### Git Commands: Working With Local Repositories

**1. git init**

* The command git init is used to create an empty Git repository.
* After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins.

**Command:** git init

### 2. git status

* The git status command tells the current state of the repository.
* The command provides the current working branch. If the files are in the staging area, but not committed, it will be shown by the git status. Also, if there are no changes, it will show the message no changes to commit, working directory clean.

**Command:** git status

The git status command shows us details about:

* **modified files** (files that are changed but not staged).
* **untracked files** (files that Git is not tracking).
* **staged files** (files that are staged and ready to be committed).

### 3. git config

* The git config command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository.
* When git config is used with --global flag, it writes the settings to all repositories on the computer.

**Command:** git config –global user.name “User-name” git config –global user.email “User-Email”

### 4. git add

* Add command is used after checking the status of the files, to add those files to the staging area.
* Before running the commit command, "git add" is used to add any new or modified files.

**Command:** git add <file-name>

git add . // to add all files

### 5. git commit

* The commit command makes sure that the changes are saved to the local repository.
* The command "git commit –m <message>" allows you to describe what has happened and help others understand.

**Command:** git commit –m “Message”

### 6. git push

* The command [git push](https://www.simplilearn.com/tutorials/git-tutorial/git-push-command) is used to transfer the commits or pushing the content from the local repository to the remote repository.
* The command is used after a local repository has been modified, and the modifications are to be shared with the remote team members.

**Command:** git push –u origin master

### 7. git branch

* The git branch command is used to determine what branch the local repository is on.
* The command enables adding and deleting a branch.

**Command:** git branch <branch-name> // Create a new branch git branch –a //List al remote or local branches git branch –d // Delete a branch

### 8. git checkout

* The git checkout command is used to switch branches, whenever the work is to be started on a different branch.
* The command works on three separate entities: files, commits, and branches. **Command:** git checkout <branch-name> // Checkout an existing branch git checkout –b <new-branch> // Checkout and create a new branch with that name.

### 9. git merge

* The [git merge](https://www.simplilearn.com/tutorials/git-tutorial/merge-conflicts-in-git) command is used to integrate the branches together. The command combines the changes from one branch to another branch.
* It is used to merge the changes in the staging branch to the stable branch.

Command: git merge <branch-name>

# Git Commands: Working With Remote Repositories

## 1. git remote

* The git remote command is used to create, view, and delete connections to other repositories.
* The connections here are not like direct links into other repositories, but as bookmarks that serve as convenient names to be used as a reference. **Command:** git remote add origin <repo-link> git remote –v // List all currently configured remote repositories.

## 2. git clone

* The git clone command is used to create a local working copy of an existing remote repository.
* The command downloads the remote repository to the computer. It is equivalent to the Git init command when working with a remote repository. **Command:** git clone <remote-URL>

## 3. git pull

* The [git pull command](https://www.simplilearn.com/tutorials/git-tutorial/git-pull-request) is used to fetch and merge changes from the remote repository to the local repository.
* The command "git pull origin master" copies all the files from the master branch of the remote repository to the local repository.

**Command:** git pull <branch-name> <remote-URL>

**4. git log** - View commit history

# Procedure

**1. Configure Git (First Time Setup)** git config --global user.name "Your Name"

git config --global user.email "your-email@example.com" git config –list

## 2. Initialize Local Repository mkdir devops-lab

cd devops-lab

git init

**3. Add Files and Commit Changes** echo "Hello DevOps" > readme.md git status git add readme.md

git commit -m "Initial commit: added readme"

## 4. View Commit History

git log git log --oneline

## 5. Push to GitHub

1. Create a repository on GitHub.
2. Connect local repo to GitHub: git remote add origin https://github.com/your-username/your-repo-name.git git branch -M main git push -u origin main

## 6. Clone Repository

git clone <https://github.com/your-username/your-repo-name.git>

## 7. Check Remote URL git remote –v

## 8. Create and Manage Branches

git branch feature-1 git checkout feature-1

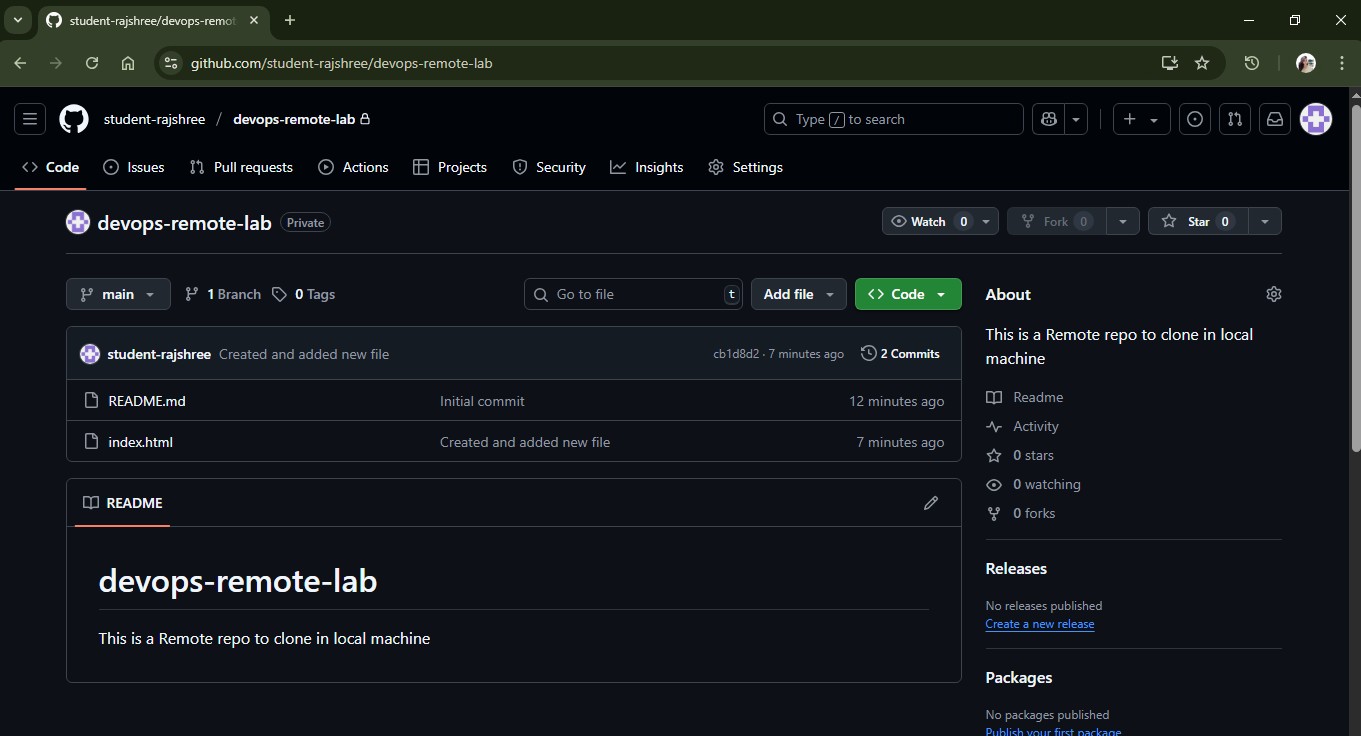
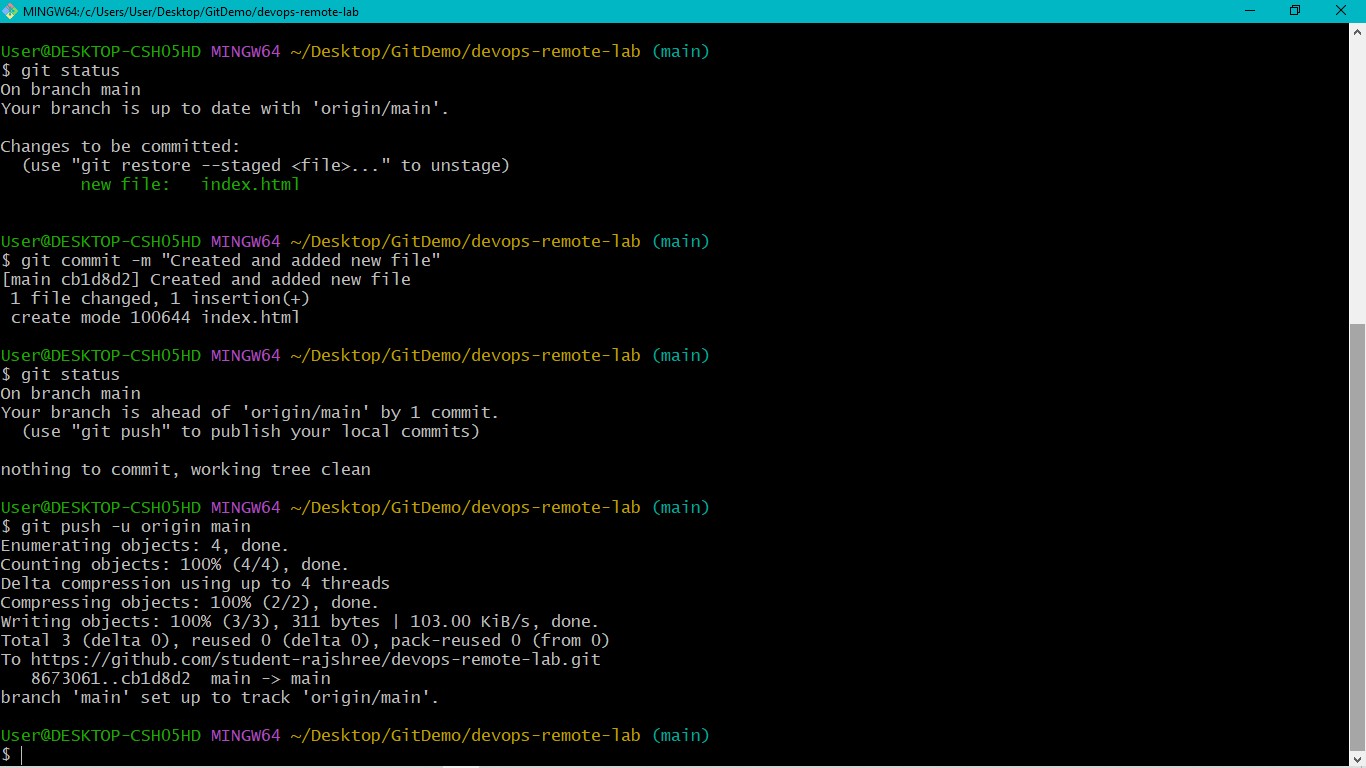
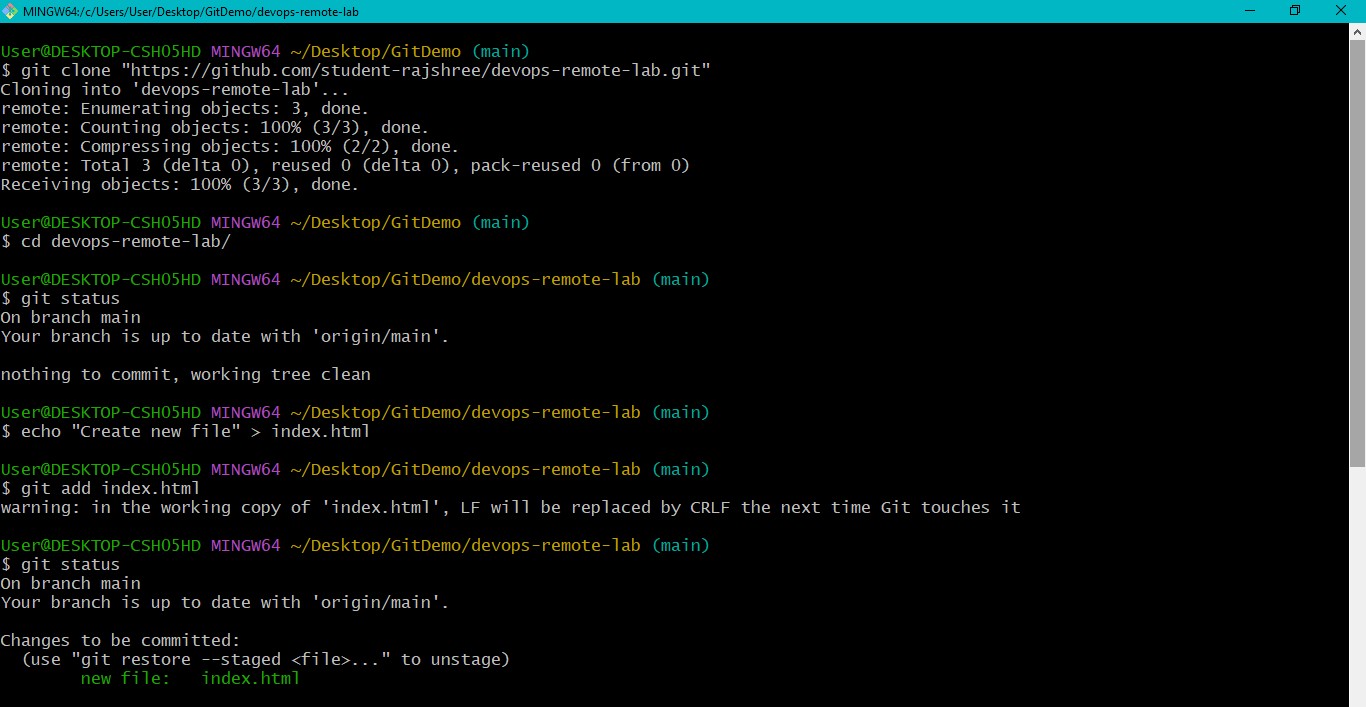
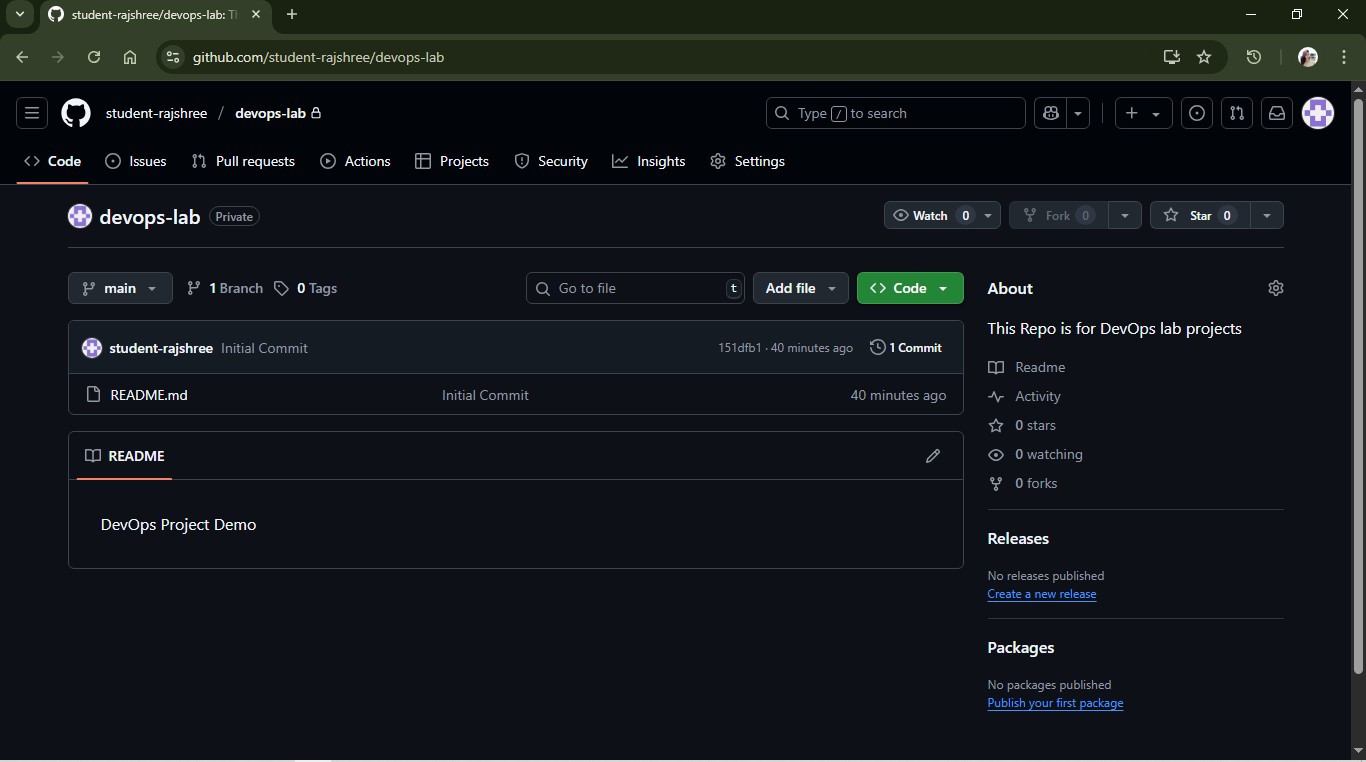
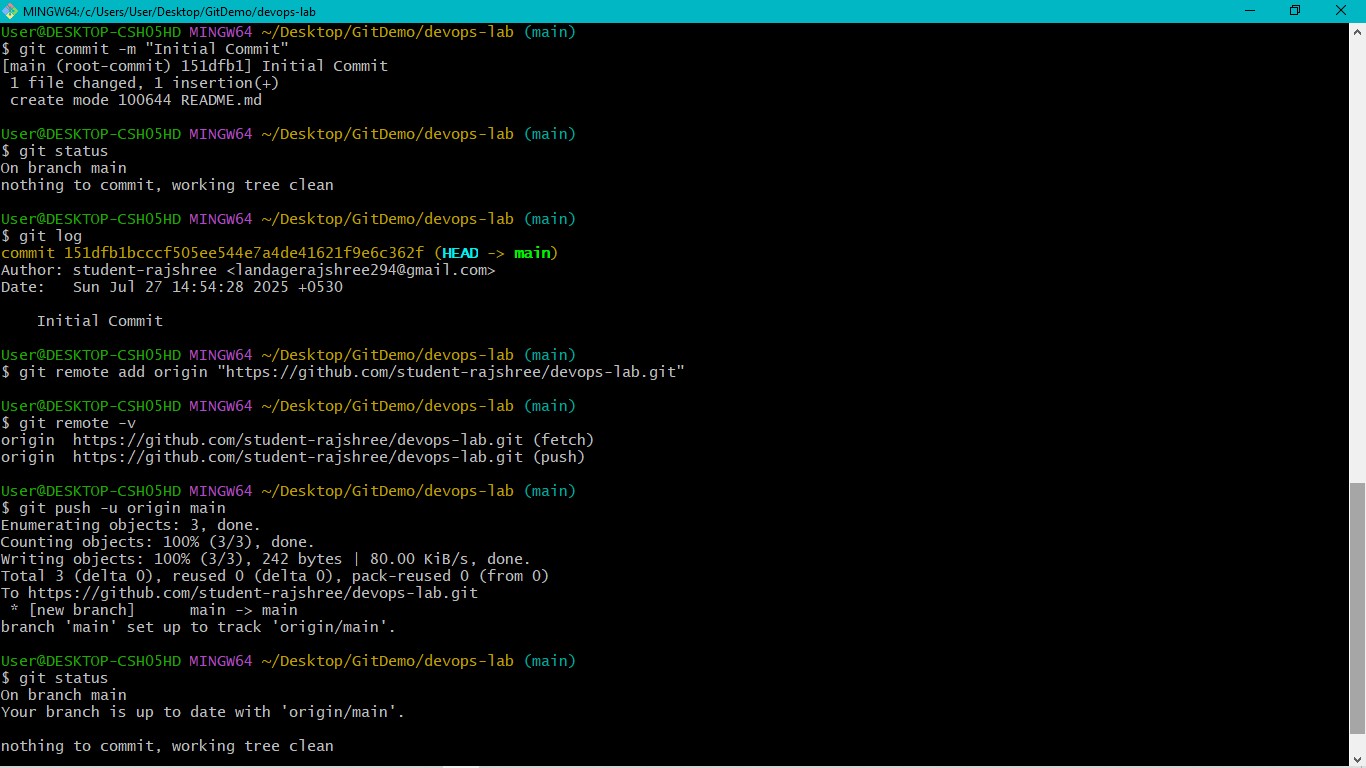
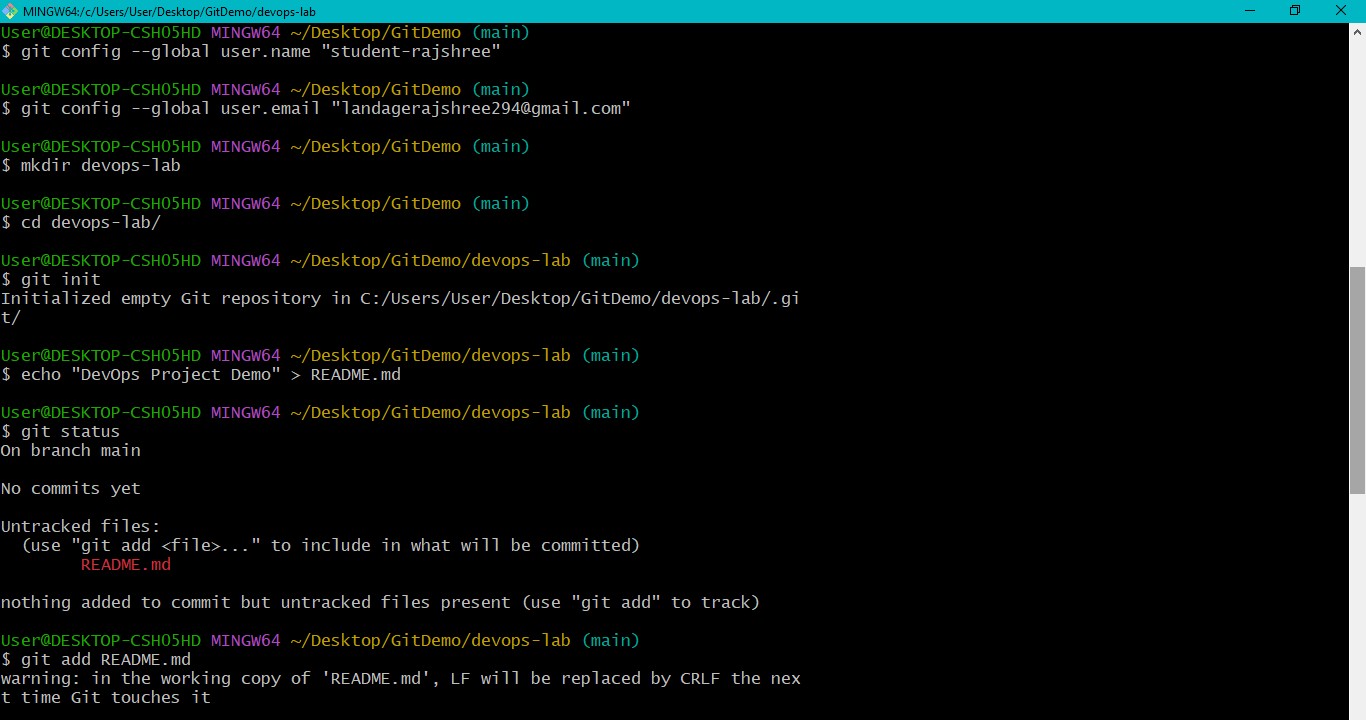
*# or using* git switch feature-1

**9. Update Files and Merge Branch** echo "Adding feature work" > readme.md git add .

git commit -m "Updated readme in feature-1 branch" git checkout main git merge feature-1

## 10. Pull Latest Changes git pull origin main

**OUTPUT**



**EXPERIMENT-03**

**AIM:** Practice Source code management on GitHub. Experiment with the source code Written in exercise 1.

**Objective:** To practice source code management activities such as adding, committing, pushing, pulling, branching, merging, and collaboration using GitHub.

**Software Requirements**

* Git installed ([https://git-scm.com/downloads)](https://git-scm.com/downloads)
* GitHub account ([https://github.com/)](https://github.com/)
* Code editor (VS Code or any)

**Procedure**

1. **Configure Git (First Time Setup)** git config --global user.name "Your Name"

git config --global user.email "your-email@example.com" git config --list

1. **Create or Clone Repository** 
   * **Create a new repository on GitHub** or
   * **Clone an existing repository:** git clone https://github.com/your-username/your-repo-name.git cd your-repo-name

1. **Initialize Local Repository (If Not Cloned)**

mkdir source-code-management

cd source-code-management

git init

1. **Add Source Code Files** echo "print('Hello DevOps')" > main.py git status git add main.py

1. **Commit Changes**

git commit -m "Added main.py with Hello DevOps script"

1. **Create and Manage Branches**

git branch feature-branch git switch feature-branch

1. **Modify Files and Commit** echo "print('Feature work added')" >> main.py git add main.py

git commit -m "Updated main.py in feature branch"

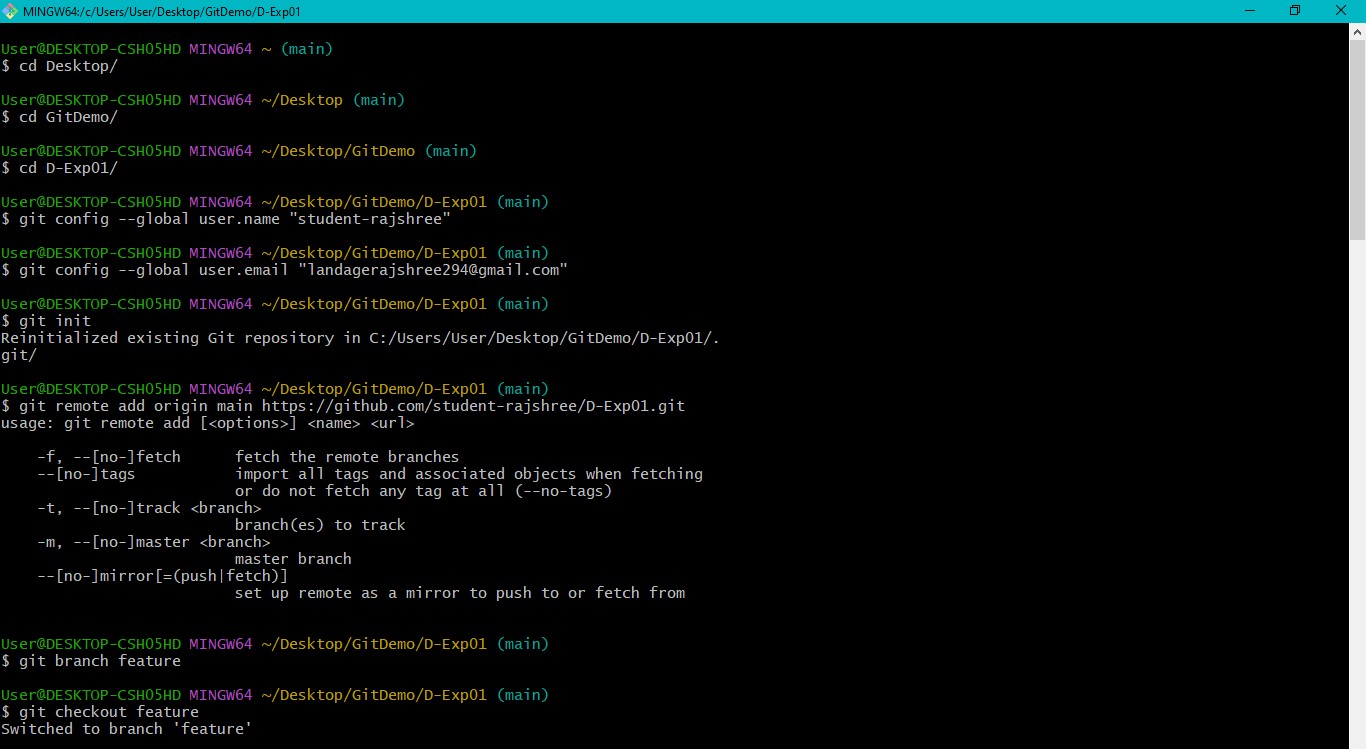
1. **Merge Feature Branch into Main Branch** git switch main git merge feature-branch

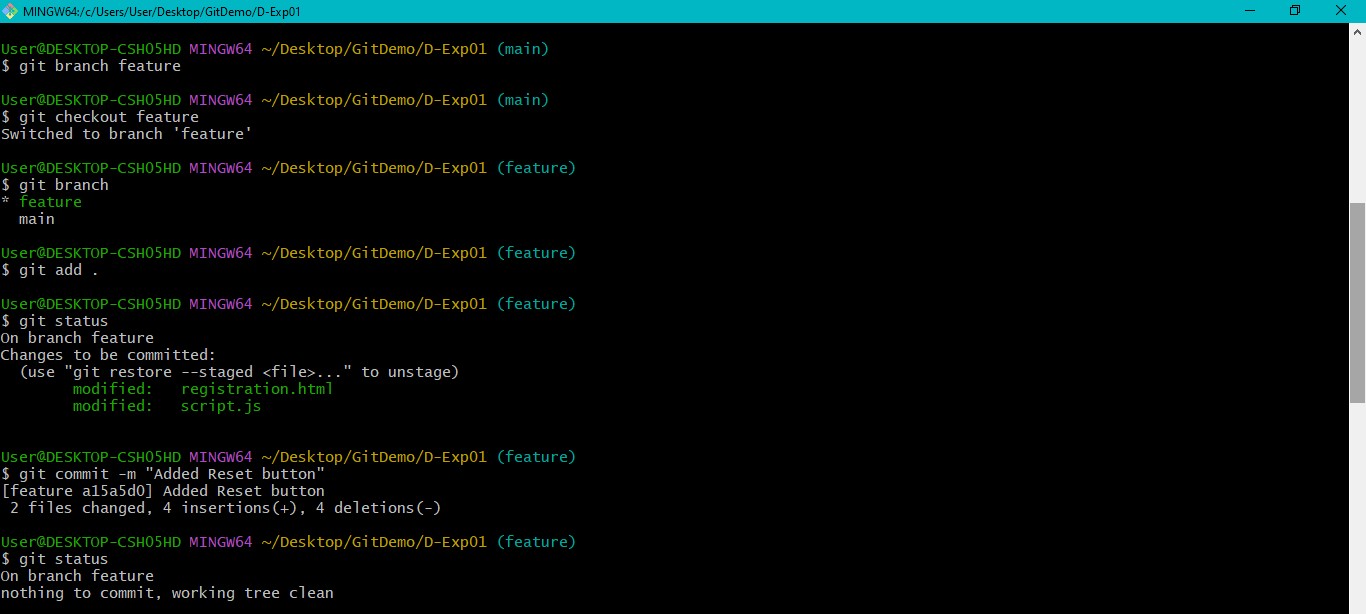
1. **Connect to GitHub and Push Code** git remote add origin https://github.com/your-username/your-repo-name.git git branch -M main git push -u origin main

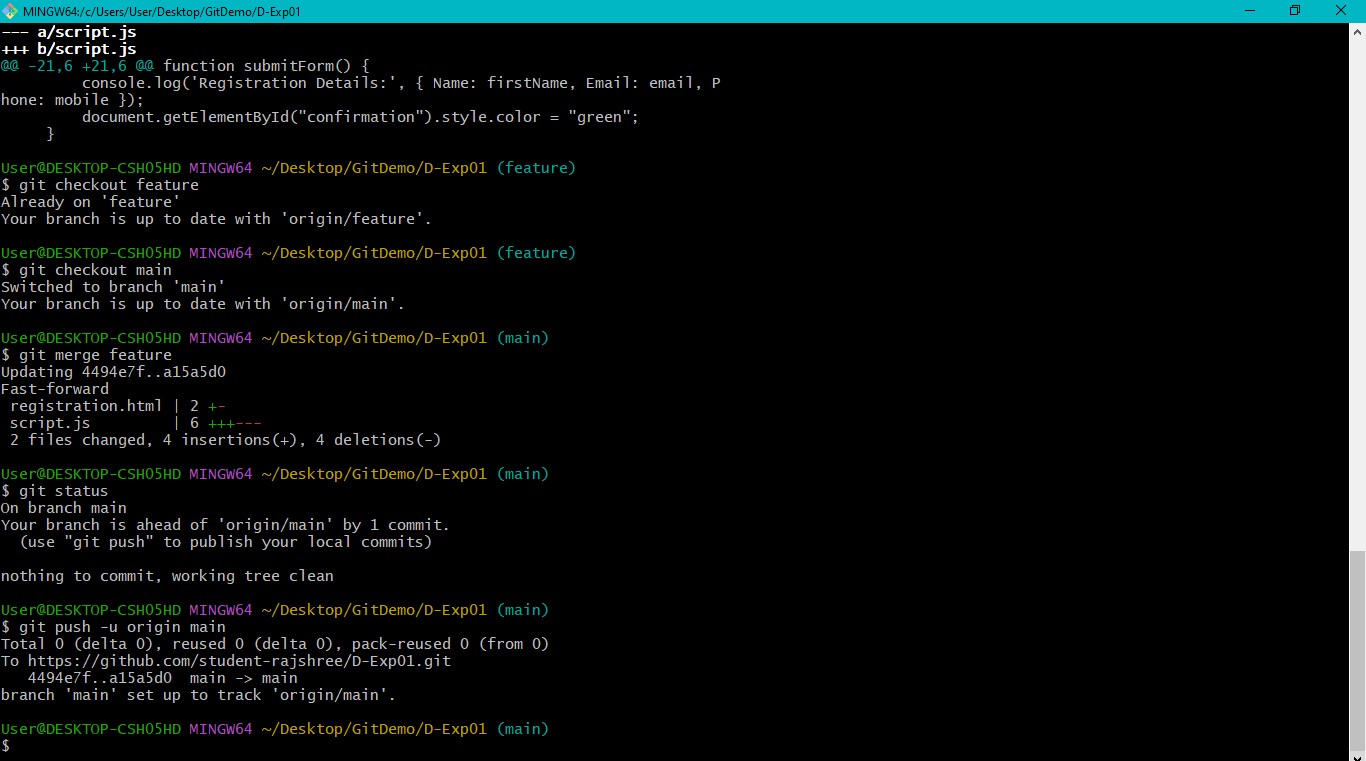
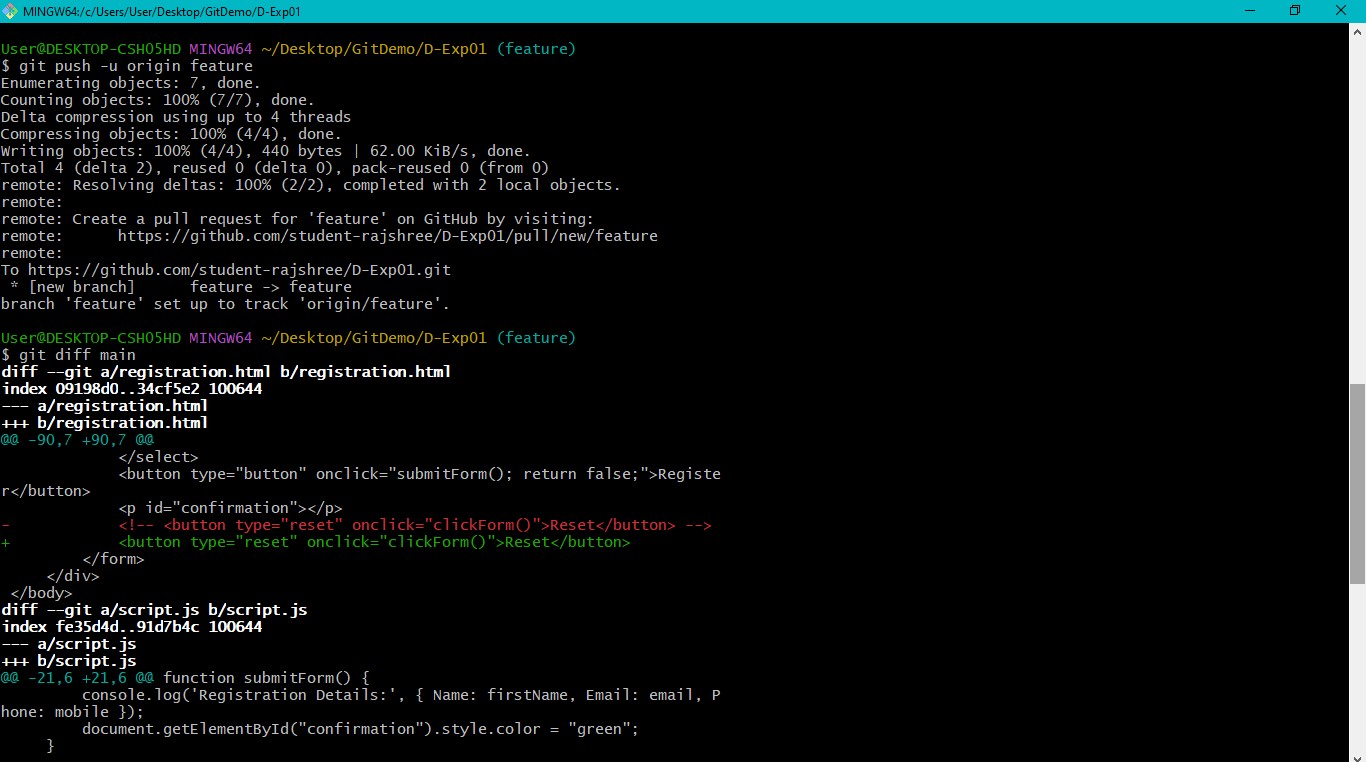
1. **Pull Updates from GitHub** git pull origin main

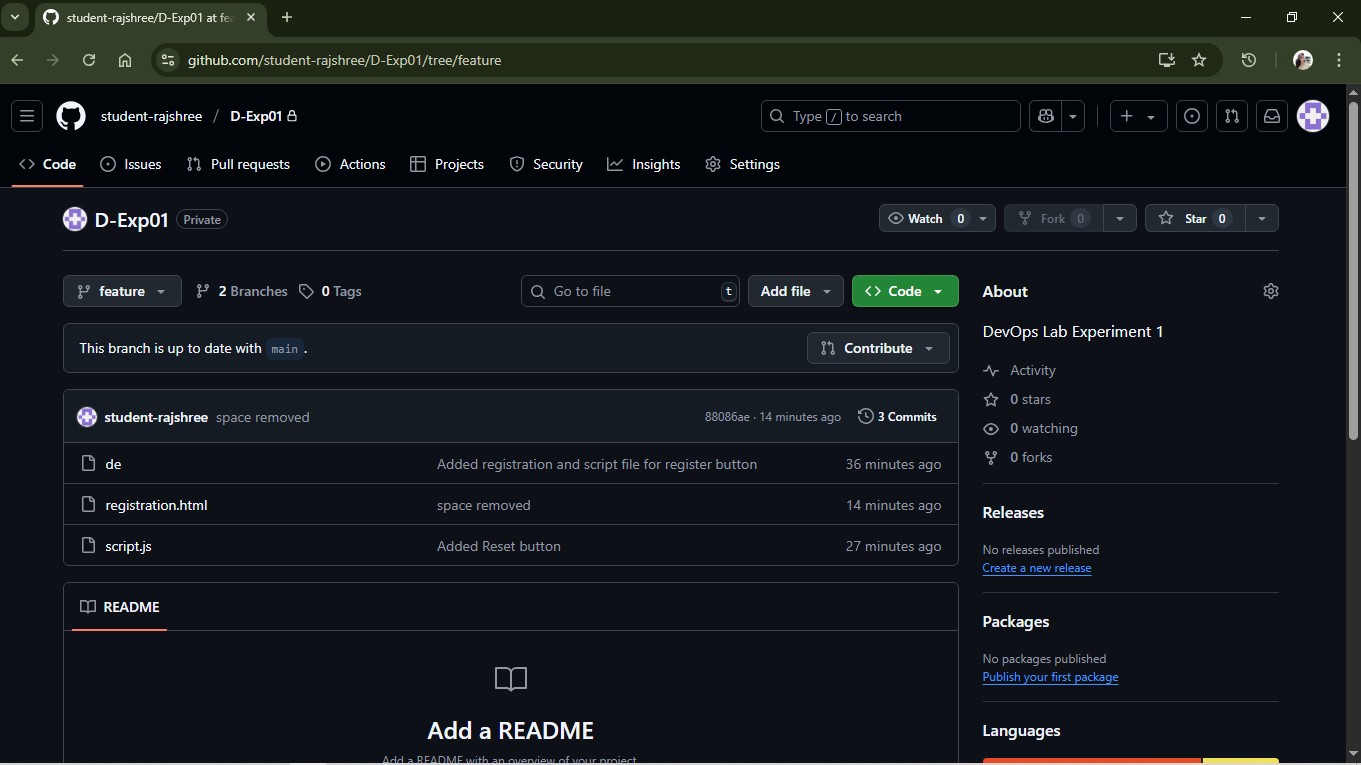
1. **Collaboration Tasks** 
   * Collaborators clone repository.
   * Create new branches.
   * Make code changes.
   * Push to GitHub.
   * Raise pull requests (PR).  Review and merge PRs.

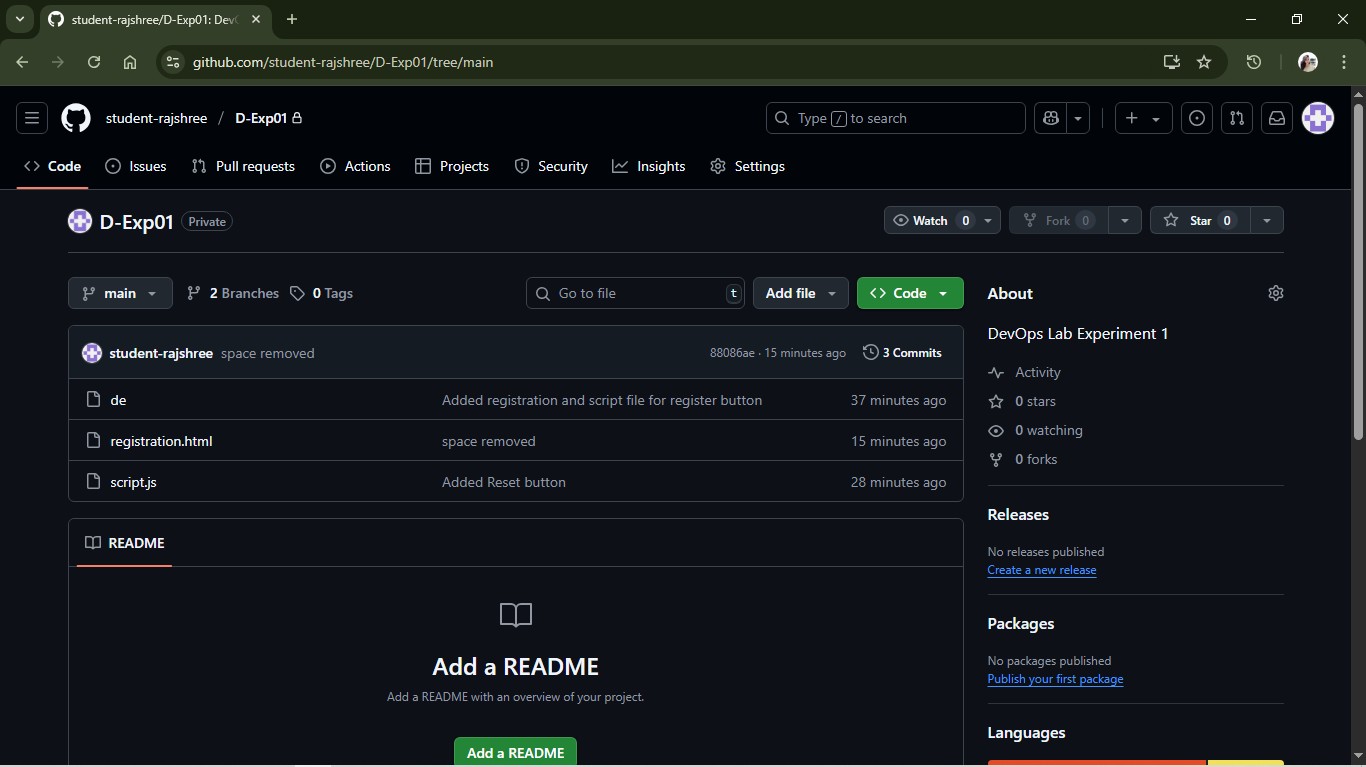
**OUTPUT**

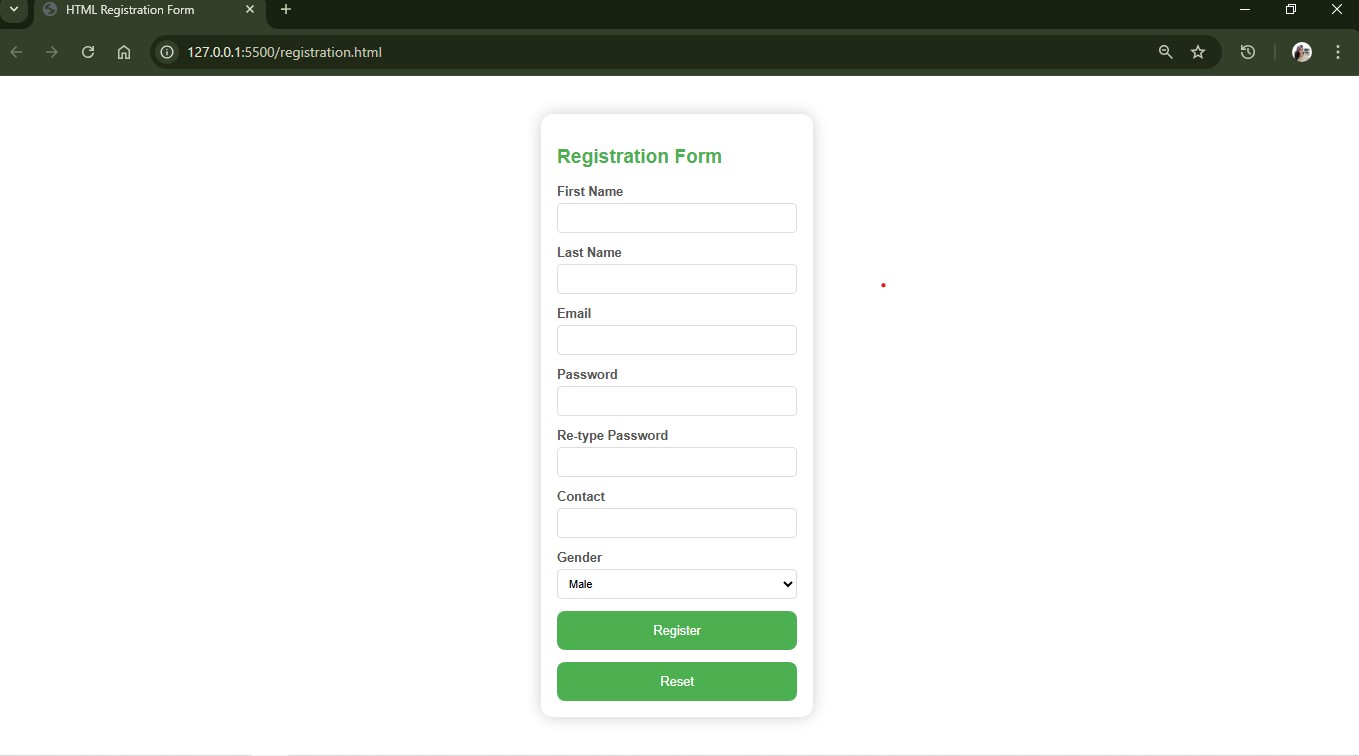






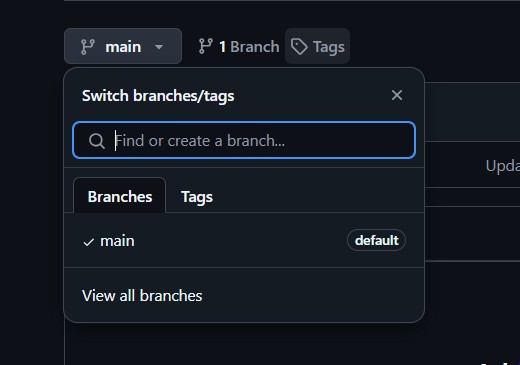




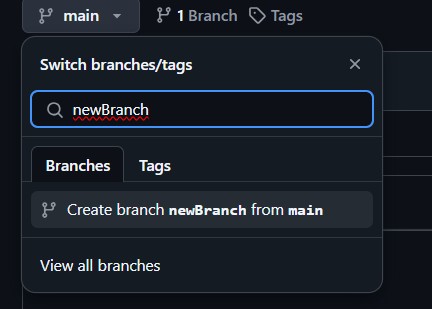


**PERFORMING BRANCHING DIRECTLY THROUGH GIT HUB**

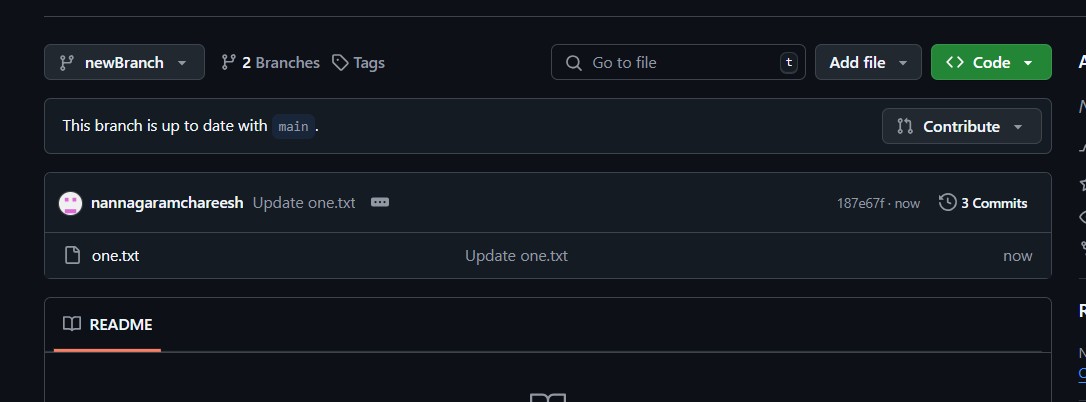
1. First go to your repository where you want to create a new branch 2. Now on the top left you will find the current branch, select it



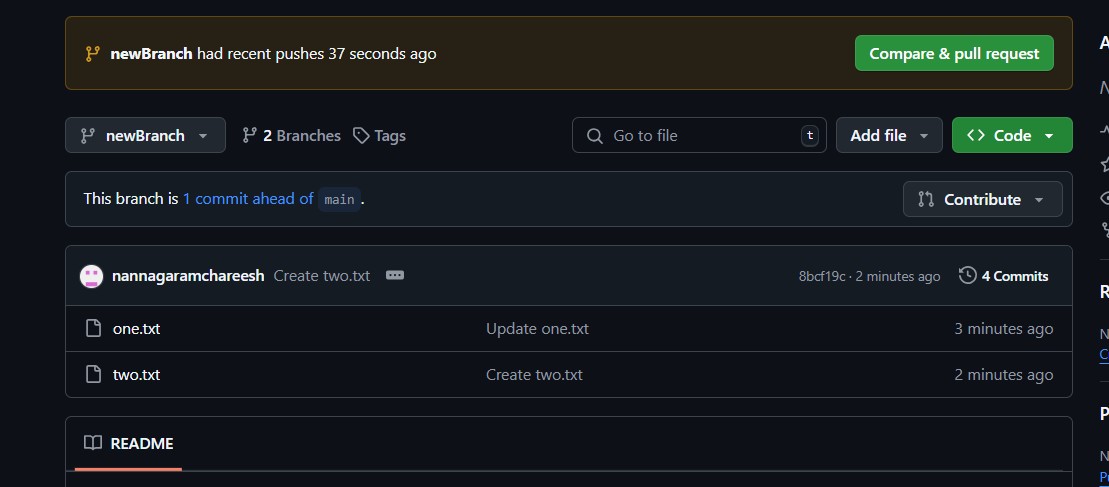
1. Now write the branch name that you want to create in the search bar. If it does not exist then you will see an option saying Create branch newBranch from main.



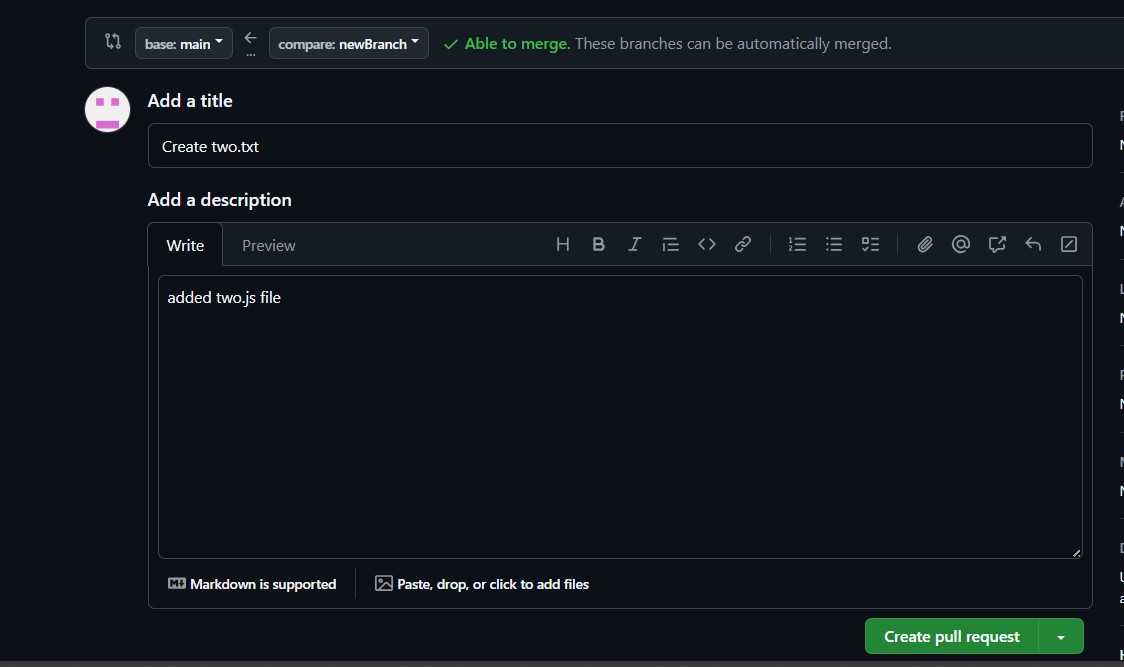
1. After clicking on Create branch, the newBranch will be created and you will automatically get switched to the new branch and all the files in the main branch will be available as a copy in this newBranch



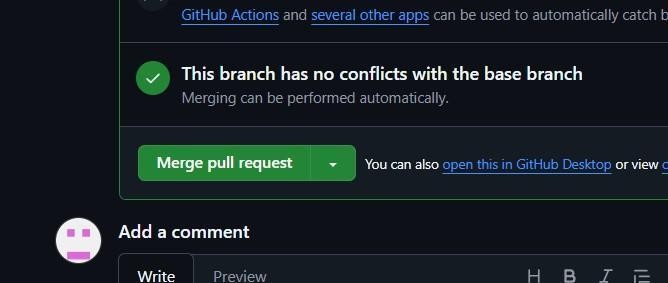
1. Create a new file in this branch write something in it and commit changes
2. Now to merge the changes in this newBranch to the main branch, click on compare & pull request



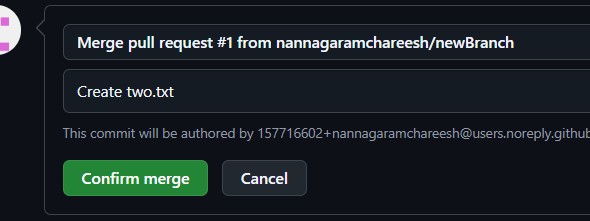
1. Now add a commit message and click on create pull request



1. Then click on Merge pull request in order to reflect the changes made in newBranch to main branch



1. Then click on confirm merge



1. After this if you go to main branch you will see that the changes made in newBranch will be reflected in the main branch So this is how branching can be done in git hub.

**CLONING A REPOSITORY**

Inorder to clone a repository use the git clone command.

1. Go ahead and copy the repository of the project that you want to clone
2. Now create a new folder and open it in vs code
3. Now open terminal in vs code and execute the command: git clone

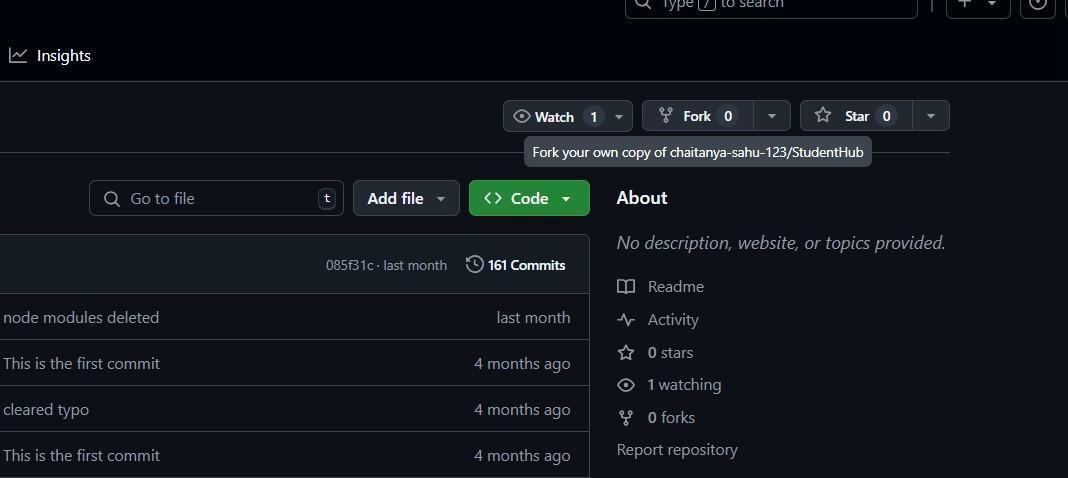
“repository\_url”

**FORKING**

Forking in GitHub is the process of cloning a repository of someone else’s to your own GitHub account where you can modify it freely.

Steps to Fork and then make changes in your local repo-

1. **Go to the repository**: Visit the GitHub page of the repository you want to fork.
2. **Click "Fork"**: On the top-right corner of the repository page, you will see a "Fork" button. Click on it.



**Experiment-04**

**AIM:** Jenkins Installation and Setup: Explore the Environment

**Objective:** To install Jenkins, perform initial setup, and explore the Jenkins environment for automation tasks.

**Software Requirements**

* Java installed (JDK 11 or higher)
* Jenkins package ([https://www.jenkins.io/download/)](https://www.jenkins.io/download/)
* Web browser

# Lab Procedure

**Install Java (If Not Installed)** sudo apt update

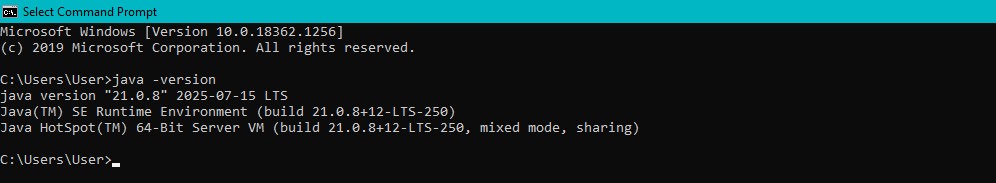
sudo apt install openjdk-11-jdk

java –version **Steps to install jdk**

* search for jdk download
* go to the first link of oracle and click jdk 21 and download x64 installer
* give the permissions and install it
* go to command prompt and click java –version to know the version of java

**Steps to install and setup**

**Jenkins**

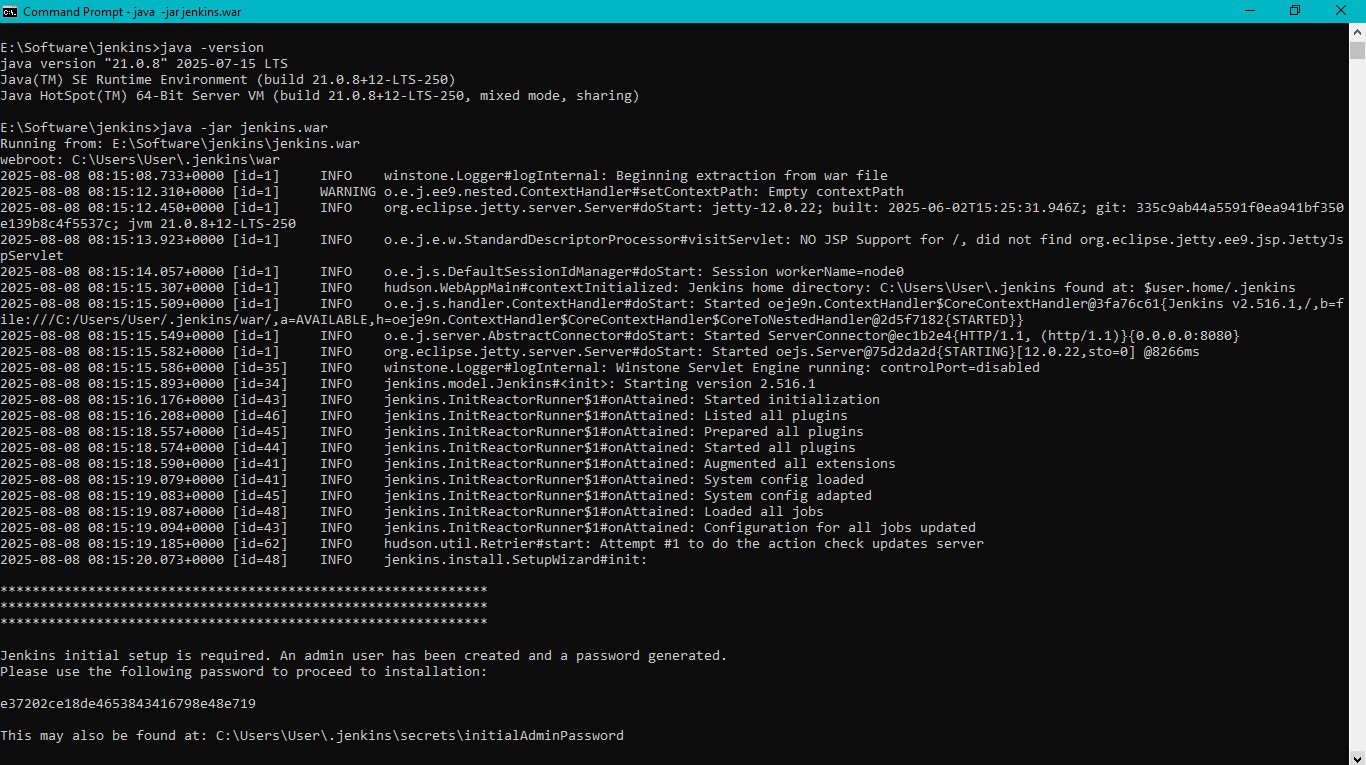


1. search for Jenkins download
2. go to the first link (Download and deploy (jenkins.io)**)**
3. now click on java generic package and download it
4. Now to run Jenkins go to command prompt and navigate to the folder where Generic java package.war (Jenkins.war) is downloaded 5. Now run this command

java

-

jar jenkins.war



6. Now a password will be generated, Copy that password

# Access Jenkins Web Interface

 Open a web browser and navigate to:

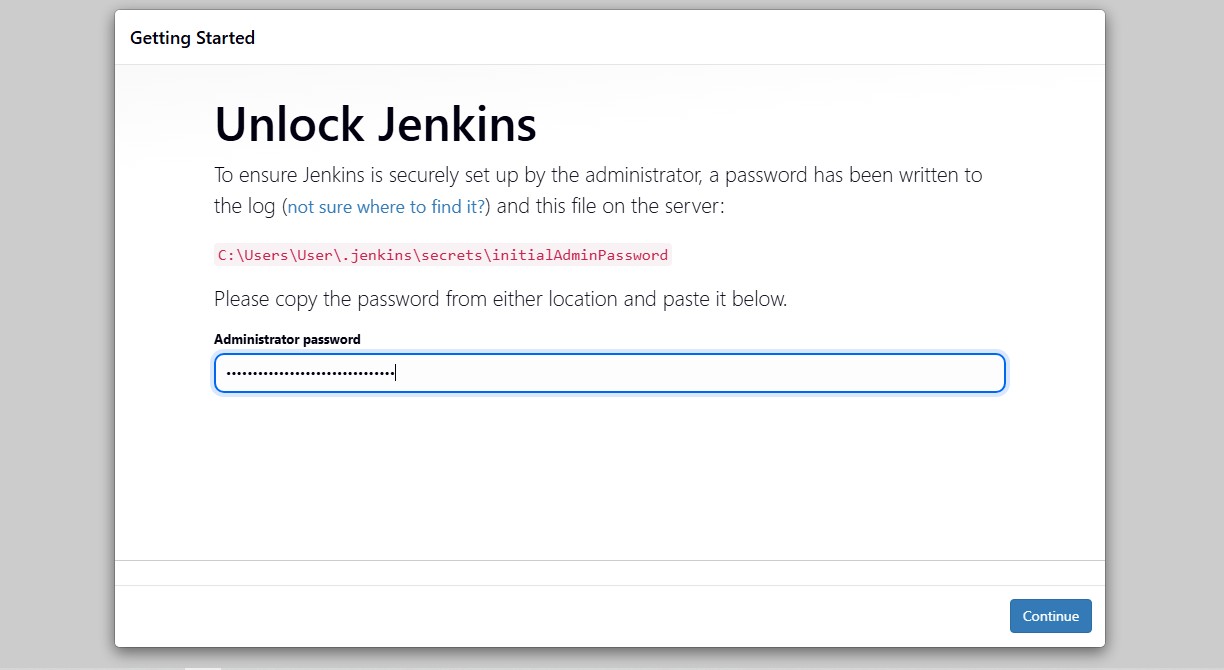
[http://localhost:8080](http://localhost:8080/)

# Unlock Jenkins

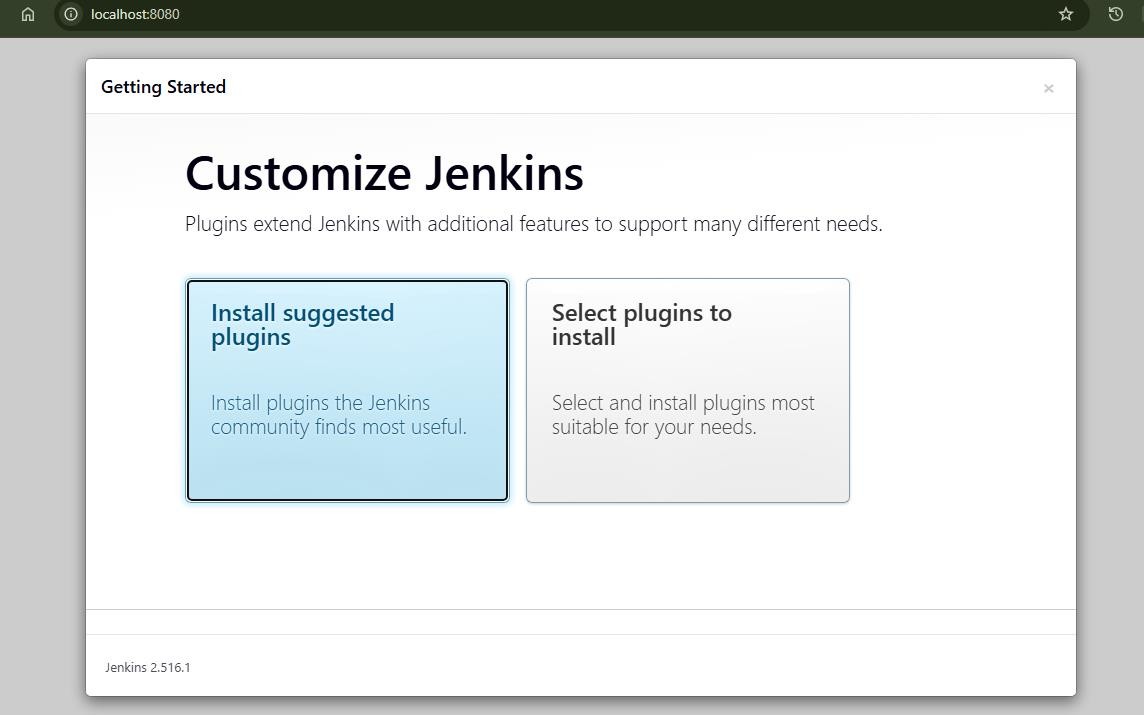
* Retrieve the administrator password:

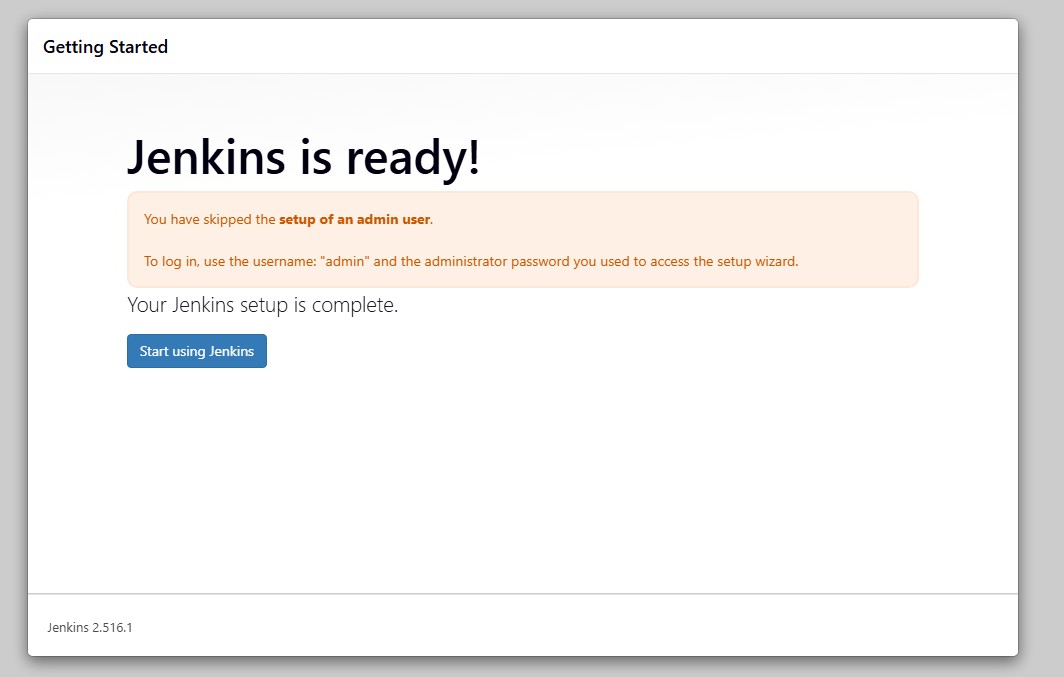
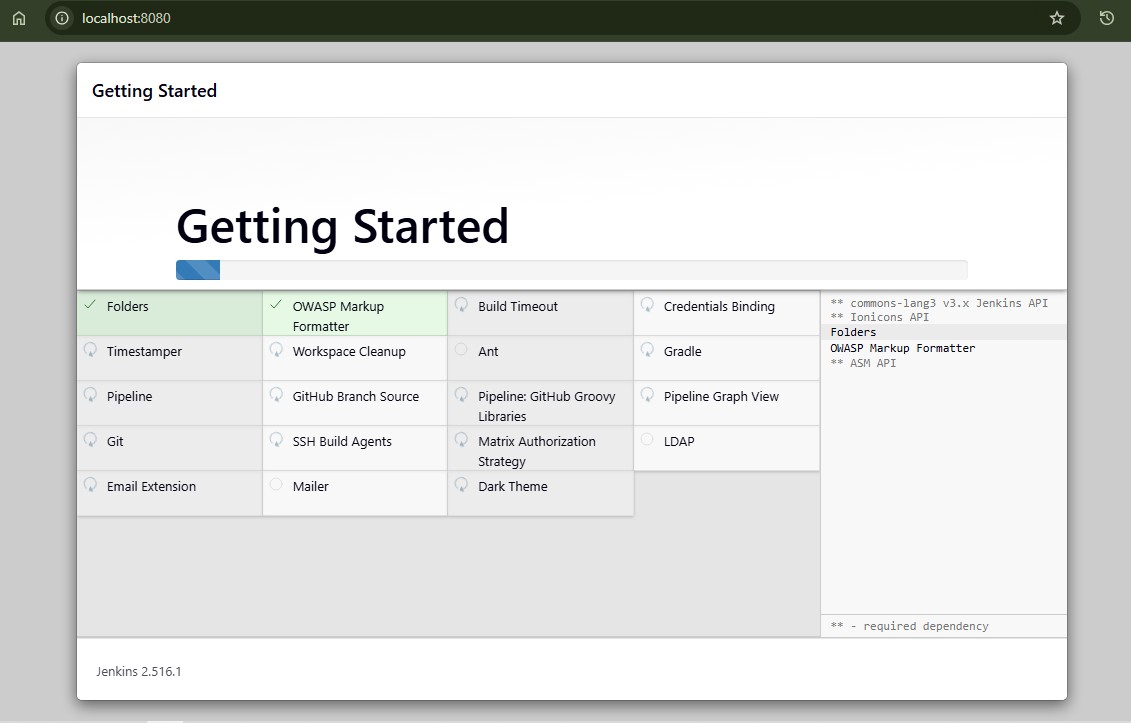
sudo cat /var/lib/jenkins/secrets/initialAdminPassword

* Paste the password into the Jenkins setup wizard.



1. **Install Suggested Plugins** 
   * Choose **Install suggested plugins** when prompted.
   * Wait for installation to complete.



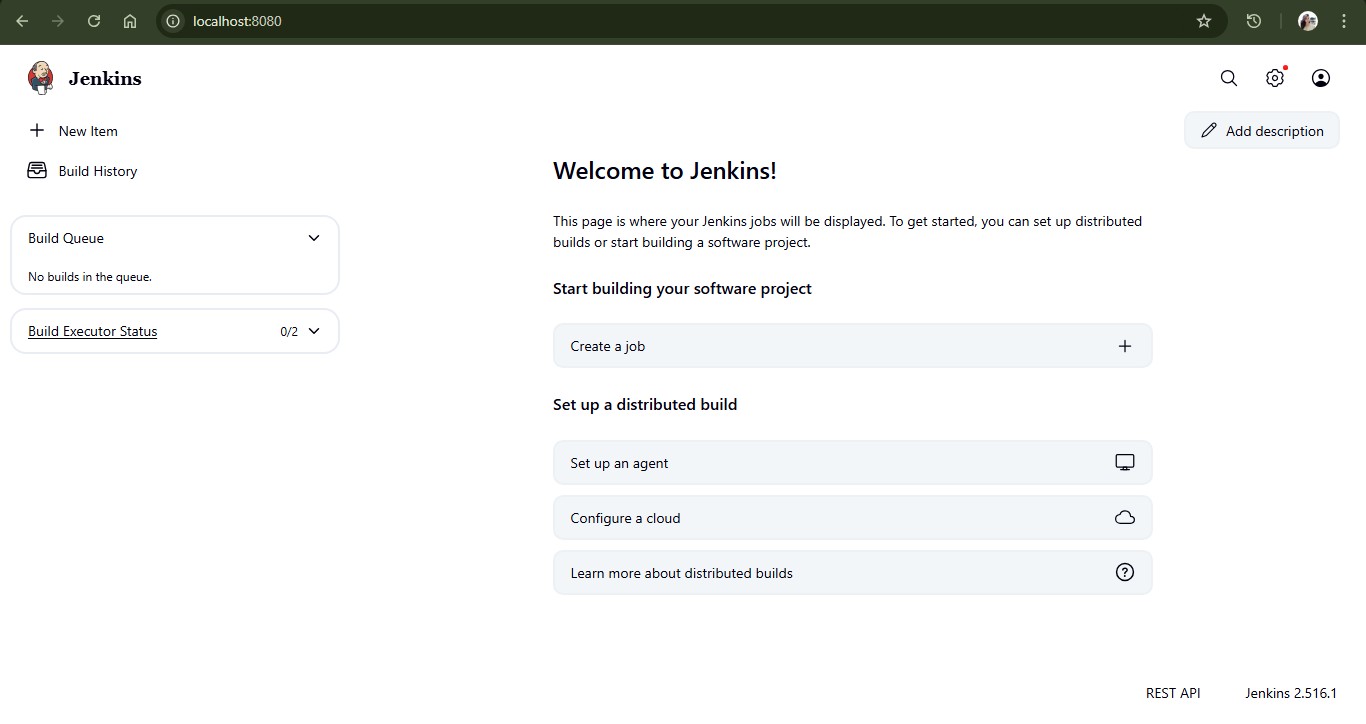


1. **Create First Admin User** 
   * Enter details like username, password, full name, and email address.

1. **Explore Jenkins Dashboard**  View Jenkins dashboard interface.
   * Explore options like **New Item**, **Manage Jenkins**, **Build History**, and **Credentials**.

**12**

**. Verify Installation**



* + Create a simple freestyle project.
  + Run the project to verify Jenkins build functionality.

# Install Jenkins

**For Ubuntu/Debian:**

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key **|** sudo apt-key add - sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.l ist.d/jenkins.list' sudo apt update sudo apt install Jenkins

**For RedHat/CentOS:**

Follow instructions from the official Jenkins website.

Start Jenkins Service sudo systemctl start jenkins sudo systemctl enable jenkins sudo systemctl status jenkins

**Conclusion**

* Students should be able to install and configure Jenkins.
* Explore the Jenkins environment for automating builds and tasks.
* Understand Jenkins dashboard and management interface.

**EXPERIMENT-05**

# AIM: Demonstrate Continuous Integration and Development Using Jenkins

**Objective:** To demonstrate continuous integration and continuous development (CI/CD) by automating build and deployment processes using Jenkins.

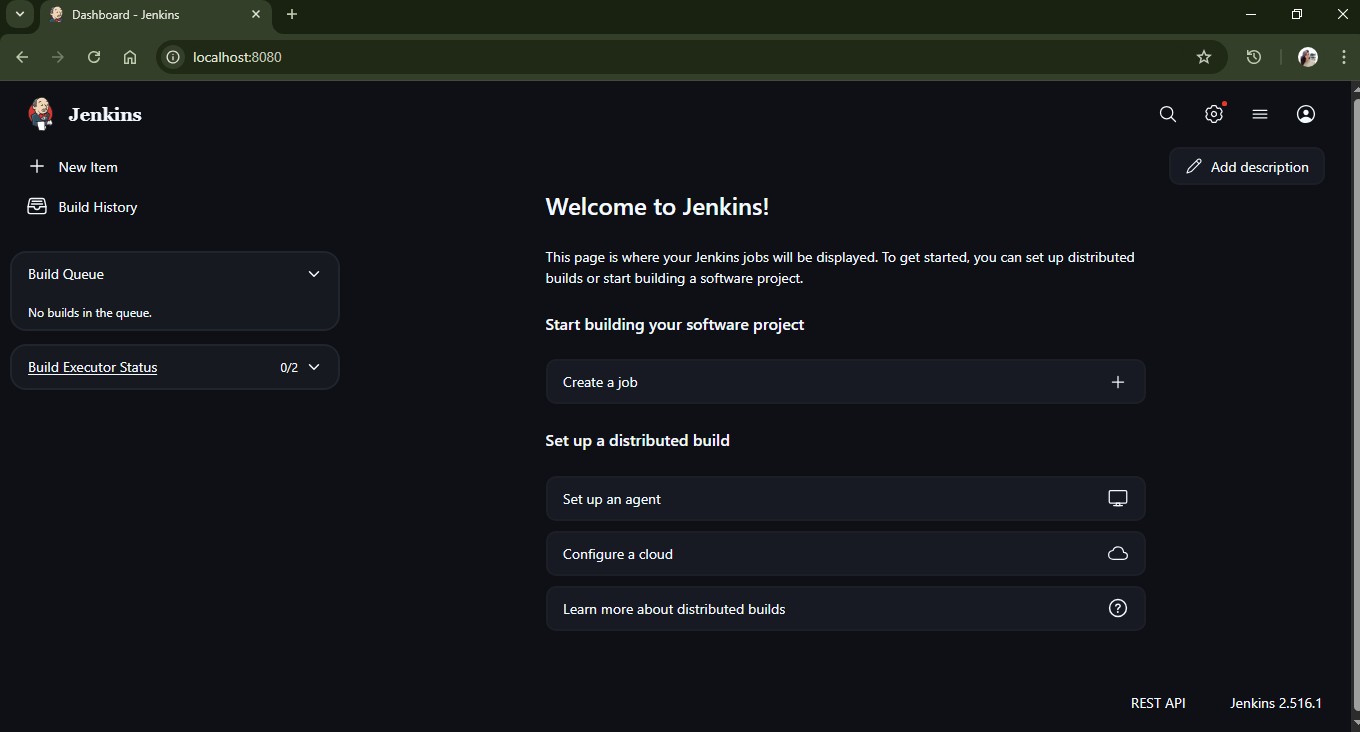
# Software Requirements

* Jenkins installed and configured ([https://www.jenkins.io/download/)](https://www.jenkins.io/download/)
* Java installed (JDK 11 or higher)
* Git installed ([https://git-scm.com/downloads)](https://git-scm.com/downloads)
* A sample project (Java/Python/Node.js etc.)
* Web browser

# Procedure

## 1. Configure Jenkins for CI/CD

 Ensure Jenkins is installed and accessible via http://localhost:8080.  Log in to Jenkins dashboard.

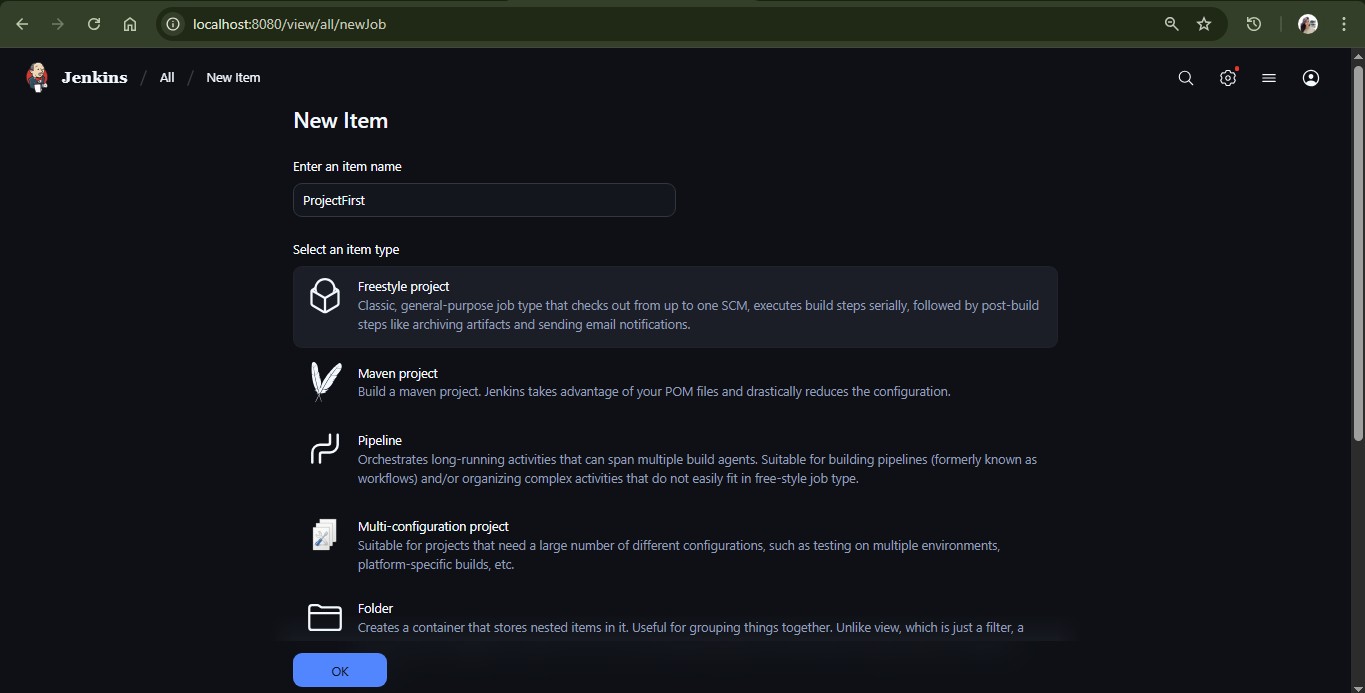


## 2. Install Required Plugins

* Go to **Settings >** **Manage Jenkins > Plugins**.
* Install the following plugins if not already installed:
  + Git Plugin
  + Pipeline Plugin
  + Any relevant build tool plugin (Maven, Gradle, etc.)

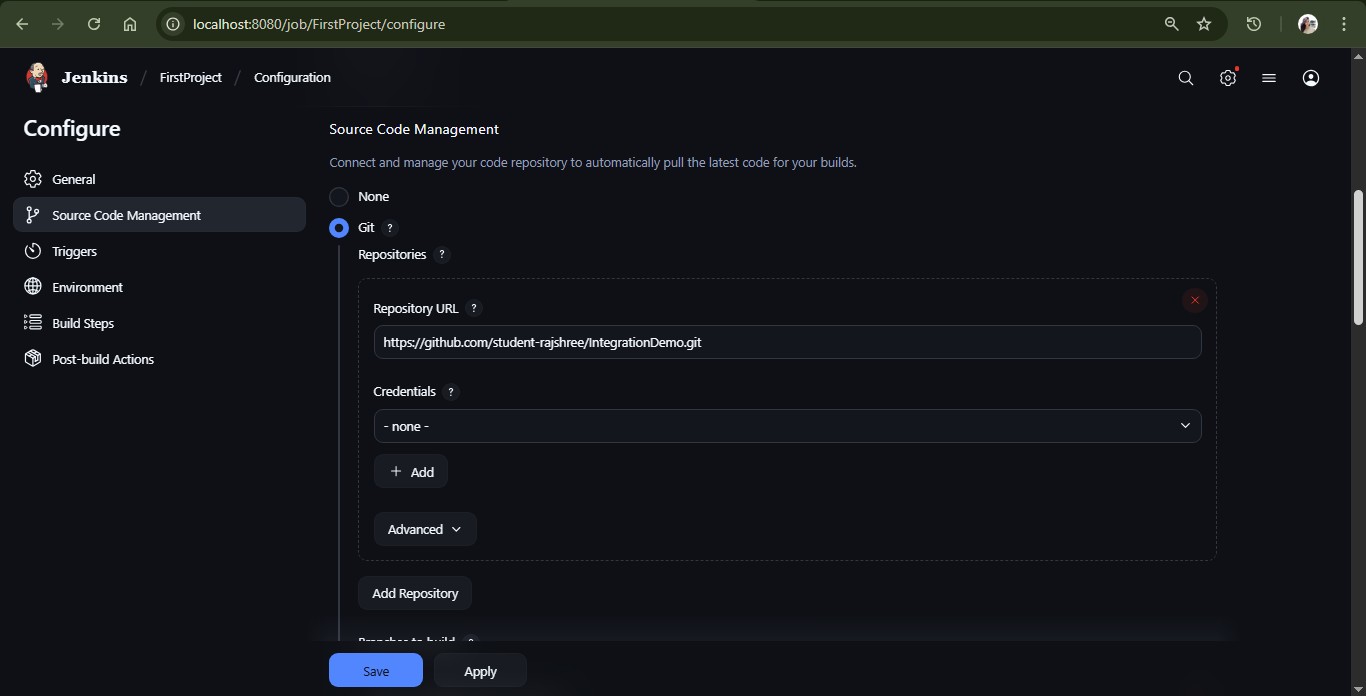
## 3. Create New Freestyle Project

* Click on **New Item**.
* Enter project name and select **Freestyle project**.  Click **OK**.



## 4. Configure Source Code Management

* In the project configuration, go to **Source Code Management**.
* Select **Git**.
* Enter repository URL (e.g., [https://github.com/your-username/sampleproject.git)](https://github.com/your-username/sample-project.git).

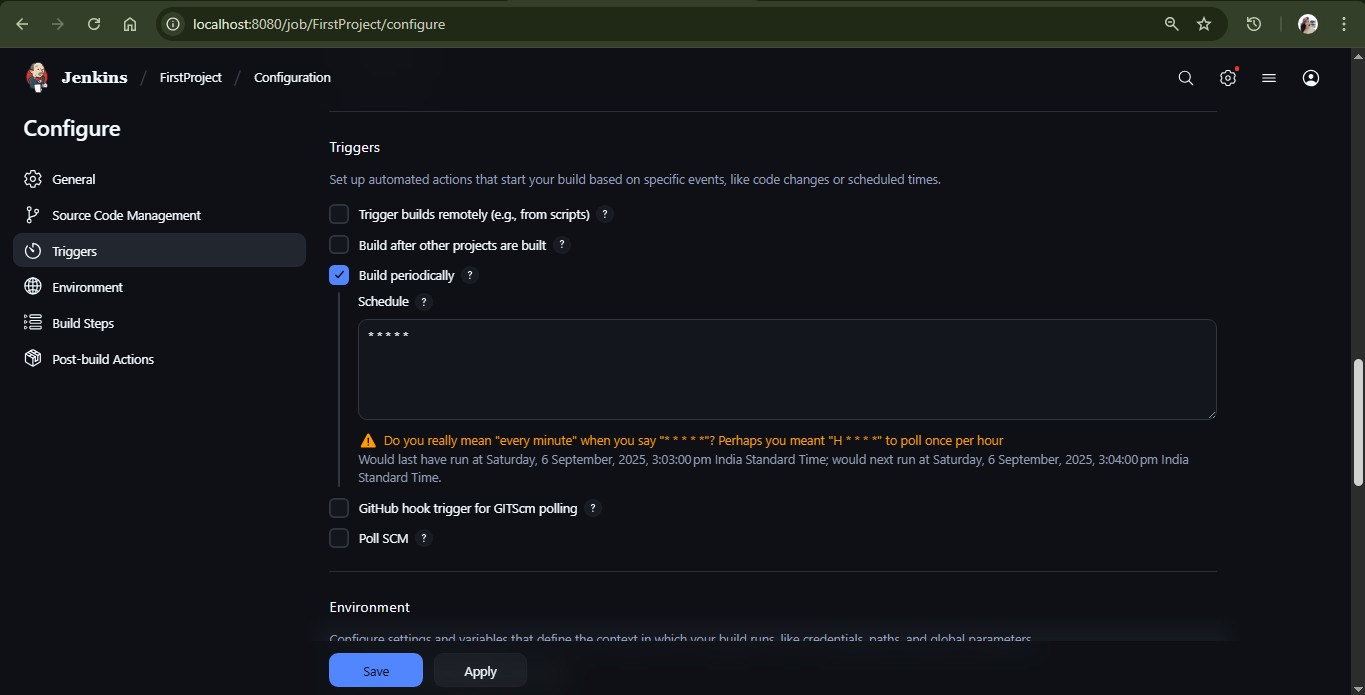


## 5. Configure Build Triggers

* Enable **Poll SCM** or **Build periodically** as needed.
* Alternatively, configure **GitHub webhook** for automatic builds.

**6**

**. Define Build Steps**

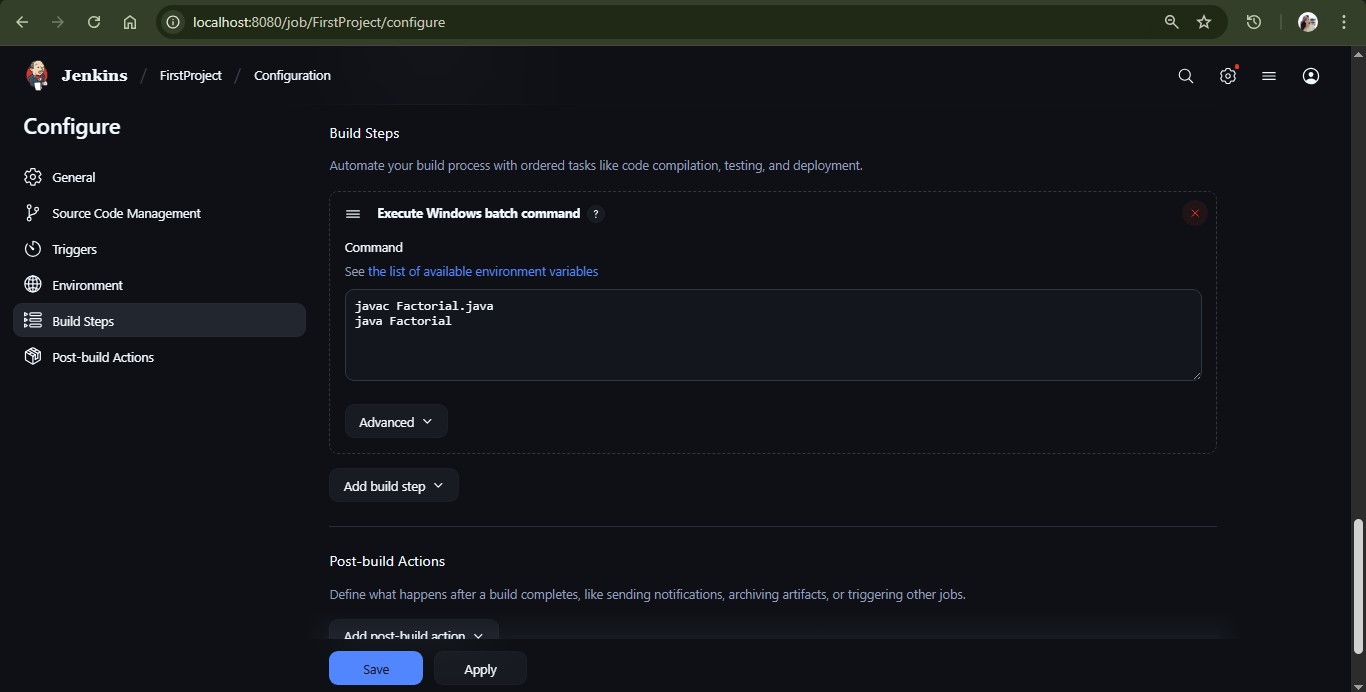


* Go to **Build** section.
* Select appropriate build step (e.g., Execute shell, Invoke Maven targets).
* Example build command:

mvn clean install

javac filename.java

java filename



**7. Optional: Post-build Actions**  Add post-build steps like:

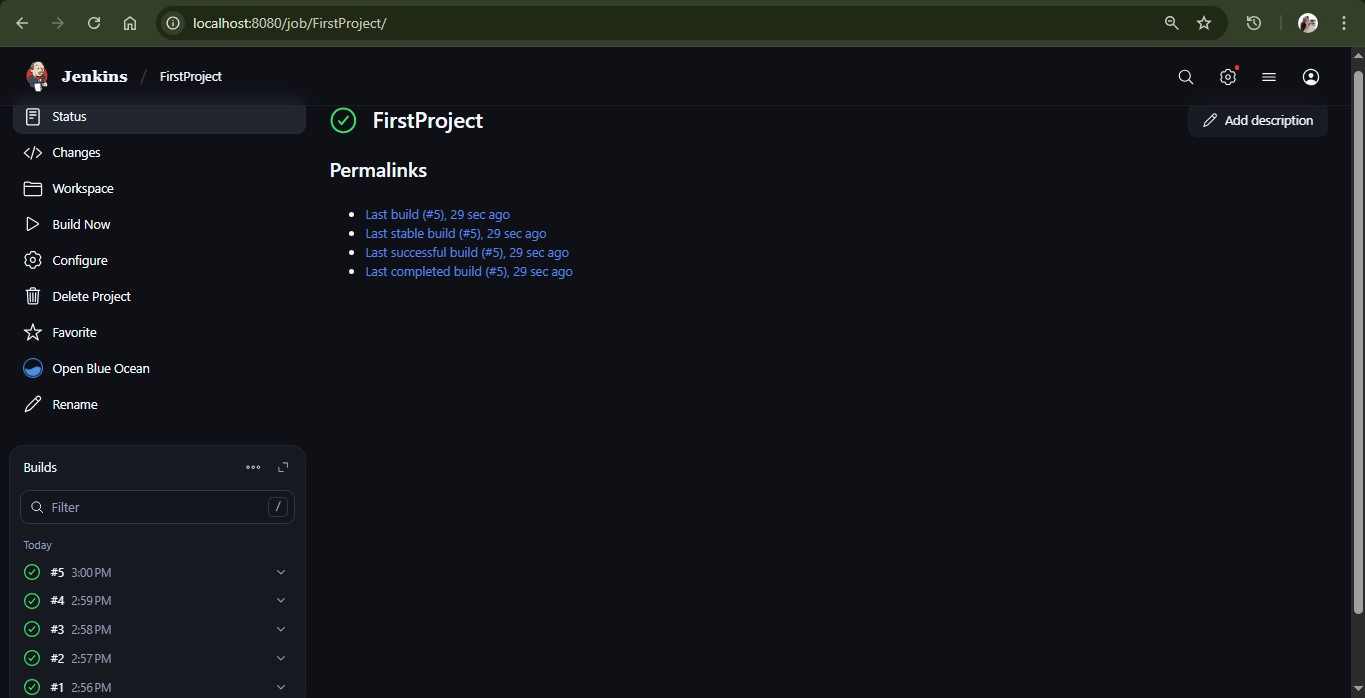
o Archive artifacts o Email notifications

## 8. Save and Build

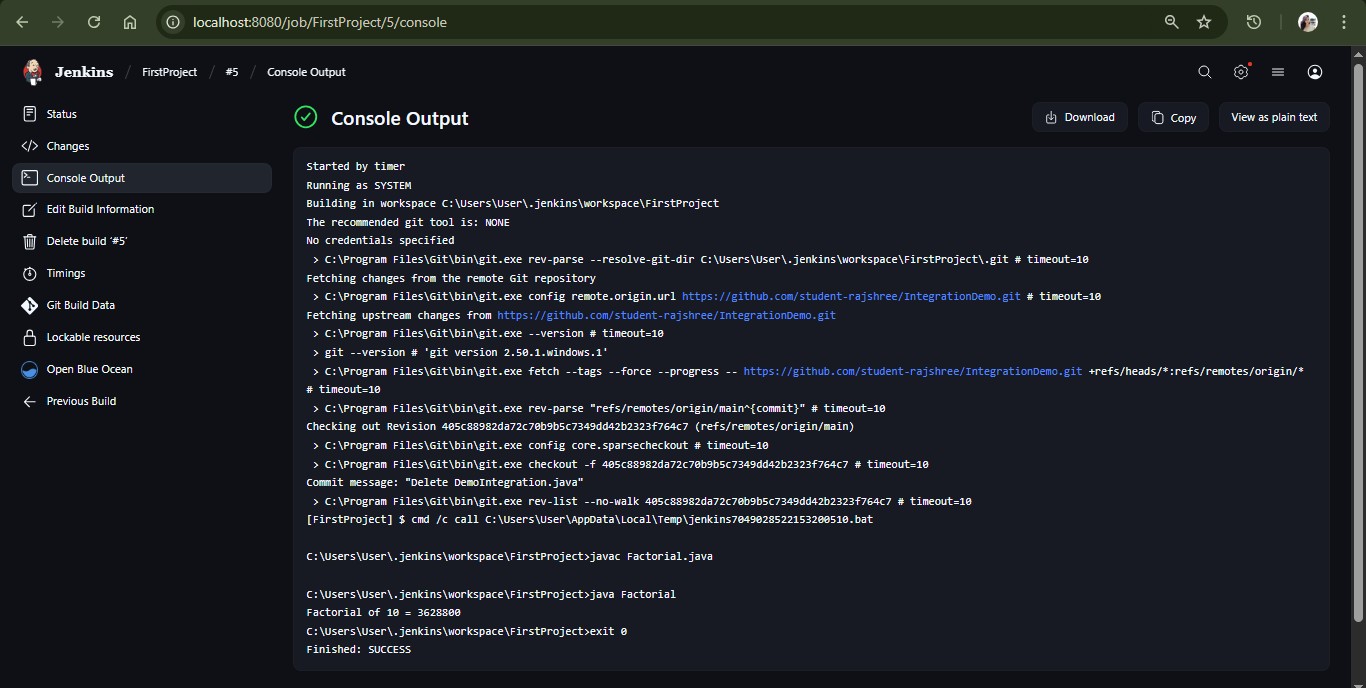
* Save the project.
* Click **Build Now**.
* Monitor the build process in **Build History**.

**9**

**. View Console Output**



* Click on build number.
* View **Console Output** to check build logs.



## 10. Automate Deployment (Optional)

 Add shell scripts or deployment tasks as build steps to automate deployment after build.

**EXPERIMENT-06**

**AIM: Explore Docker commands for content management.**

**Objective:** To explore and understand Docker commands for managing images, containers, files, and volumes (content management).

# Software Requirements

* **Docker installed and configured** (Docker Desktop for Windows/macOS or Docker Engine for Linux) –<https://www.docker.com/get-started>
* **Operating System**: Linux (Ubuntu 20.04 or later) / Windows 10 or later with WSL2 / macOS (latest)
* **Command-line terminal** (Bash / PowerShell) for executing Docker commands
* **Internet connectivity** to pull images from Docker Hub or other registries
* **Optional text editor** (VS Code) to create and edit files inside containers

**Docker** is an open-source platform that allows developers to easily build, deploy, and run applications in containers.

Docker allows developers to create a container image that includes all the dependencies needed for an application to run, regardless of the underlying operating system or infrastructure.

Docker also provides tools for managing container images, such as Docker Hub, a public registry of images, and Docker Compose, a tool for defining and running multi-container applications.

# Docker Commands

1. **docker**: This command is used to list all docker commands.
2. **docker –version:** This command is used to check docker version.
3. **docker pull ImageName:** This command is used to download a Docker image from a registry or from docker hub.
4. **docker run ImageName:** This command is used to create a new Docker container from an image.
   1. **docker run –d ImageName:** This command is used to build a container in background.
   2. **docker run ImageName: Version:** This command is used to pull and create a specific version of an image.
   3. **docker run –it ImageName:** This command is used to access terminal of that container in our local machine in the interactive mode.
   4. **docker run –name containerName –d ImageName:** This command is used to give custom name to a container.
5. **docker images:** This command is used to list all the Docker images that are currently available on your system.
6. **docker rm ContainerName:** This command is used to remove a Docker container.
7. **docker rmi ImageName:** This command is used to remove an docker image.
8. **docker start ContainerName or ContainerID:** This command is used to restart an existing container.
9. **docker stop ContainerName or ContainerID:** This command is used to stop a running container.
10. **docker ps:** This command is used to list all the running Docker containers.
11. **docker ps –a:** This command is used to list all the Docker containers, including running and stopped containers.
12. **docker logs ContainerID or ContainerName:** This command is used to check the logs of the container which can help to identify the root cause of the problem.
13. **docker exec –it ContainerID or ContainerName: bash/terminal/sh** This command is used to execute additional commands in a running container. We can access that containers bash/terminal.
14. **docker compose –f FileName.yaml up -d:** This command is used to manage multi-container Docker applications.

**-f:** Option for file name **up –**d: Whenever the containers are defined inside the yaml file and if we want to create and start them in the detached mode we will choose up.

1. **docker compose –f FileName.yaml down:** Whenever the containers are defined inside the yaml file and if we want to delete them then we will choose down.
2. **docker build –t ImageName: version .** : This command is used to build docker image.

**-t or—tag:** Version of images **dot (.): Build context** (Current Directory)

1. **docker login –u UserName:** The Docker login command will help you to authenticate with the Docker hub by which youcan push and pull your images.
2. **docker push Image Name:** This command is used to push docker image to dockerhub.
3. **docker volume create VolumeName:** This command is used to create a volume.
4. **docker volume ls:** This command is used to list all volumes available.
5. **Docker run –it –v VolumeName: ContainerPath:** This command is used to attach the volume to a path inside the container.
6. **docker volume rm VolumeName:** This command is used to remove the volume.
7. **docker volume prune:** This command is used to remove unused volumes.

# Some important flags

**-d:** To run any container in background.

**-e:** To set environment variables

**-p:** To bind the host port with the container port

**EXPERIMENT-07**

**AIM: Develop a simple containerized application using Docker.**

**Objective:** To understand how to create, containerize, and run a simple application using Docker.

# Software Requirements

* **Docker installed and configured** (Docker Desktop for Windows/macOS or Docker Engine for Linux) –<https://www.docker.com/get-started>
* **Operating System**: Linux (Ubuntu 20.04 or later) / Windows 10 or later with WSL2 / macOS (latest)
* **Command-line terminal** (Bash / PowerShell) for executing Docker commands
* **Git** for source control
* **Internet connectivity** to pull images from Docker Hub or other registries
* **Optional text editor** (VS Code) to create and edit files inside containers

# Lab Procedure

1. Clone the [getting-started-app](https://github.com/docker/getting-started-app) repository using following command:

git clone <https://github.com/docker/getting-started-app.git>

1. View the contents of the cloned repository. You should see the following files and sub-directories.

├── getting-started-app/

│ ├── .dockerignore

│ ├── package.json

│ ├── README.md

│ ├── spec/

│ ├── src/

│ └── yarn.lock

1. Create a Docker file in the same directory and add the following code

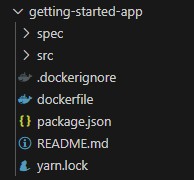
FROM node:18-alpine

WORKDIR /app COPY . .

RUN yarn install --production

CMD ["node", "src/index.js"]

EXPOSE 3000

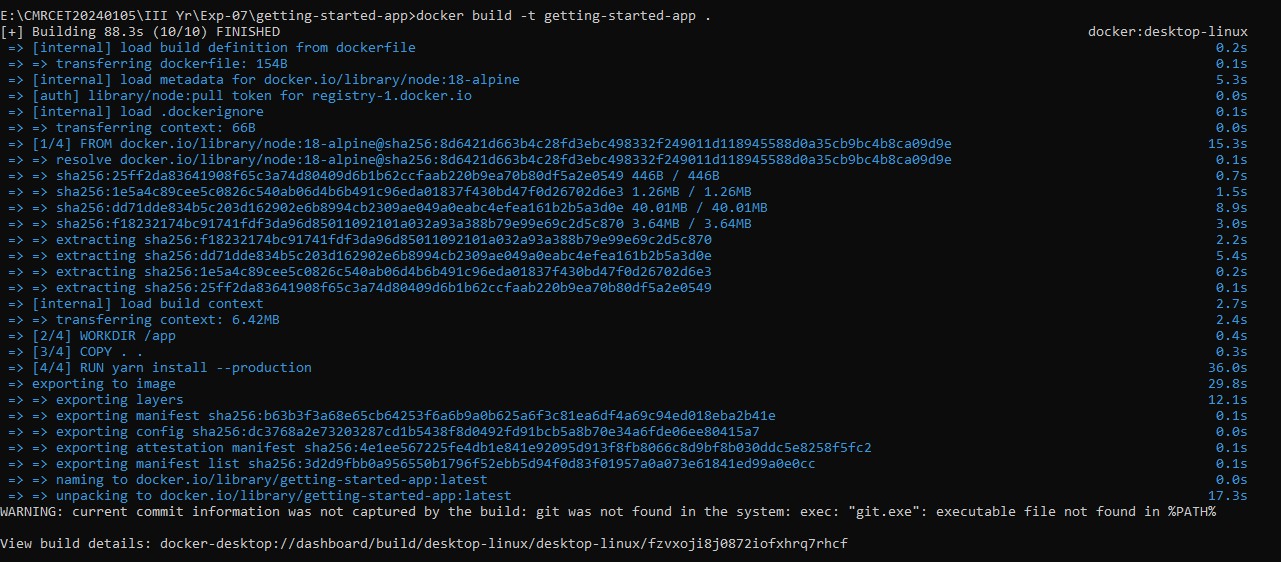


In the terminal, make sure you're in the getting-started-app directory. Replace /path/to/getting-started-app with the path to your getting-started-app directory.

*cd /path/to/getting-started-app*

1. **Build the image**

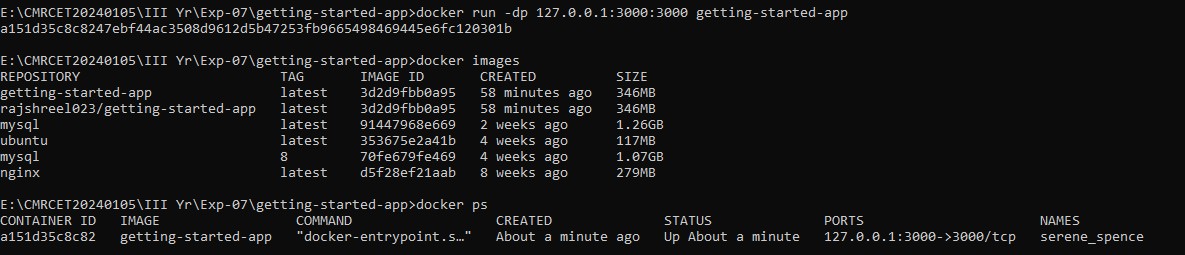
Run the following command in the terminal **docker build –t getting-started-app .**



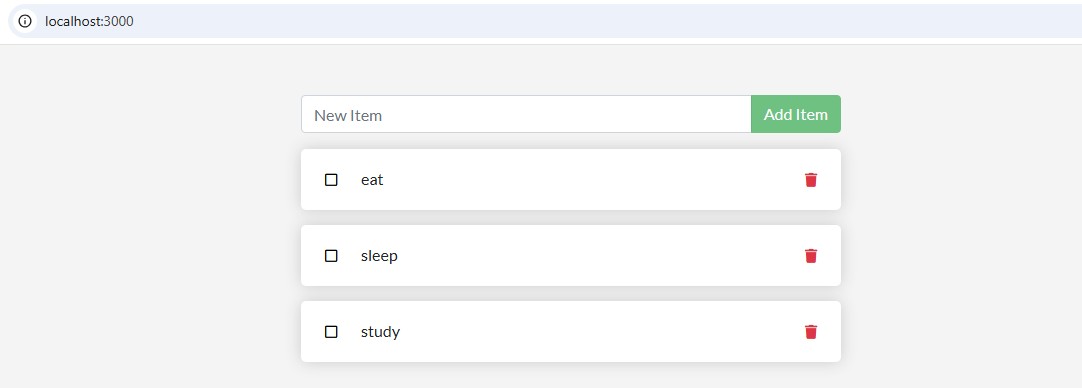
1. Verify the image is created or not by using **docker images** command



1. Run the image using following command docker run -dp 127.0.0.1:3000:3000 getting-started-app



1. Open browser and type https://localhost:3000 to check the output the image is running on port 3000



**EXPERIMENT-08**

**AIM: Integrate Kubernetes and Docker**

**Objective:** To integrate Kubernetes with Docker for orchestrating containerized applications using Pods, Deployments, and Services.

**Software Requirements**

* Docker installed ([https://www.docker.com/products/docker-desktop/)](https://www.docker.com/products/docker-desktop/)
* Kubernetes installed (Minikube or Docker Desktop with Kubernetes enabled)
* kubectl command-line tool
* Code editor (VS Code or any)
* Sample containerized application

The Kubernetes server runs as a single or multi-node cluster, within Docker container(s). This lightweight setup helps you explore Kubernetes features, test workloads, and work with container orchestration in parallel with other Docker functionalities.

Docker Desktop includes a standalone Kubernetes server and client, as well as Docker CLI integration that runs on your machine.

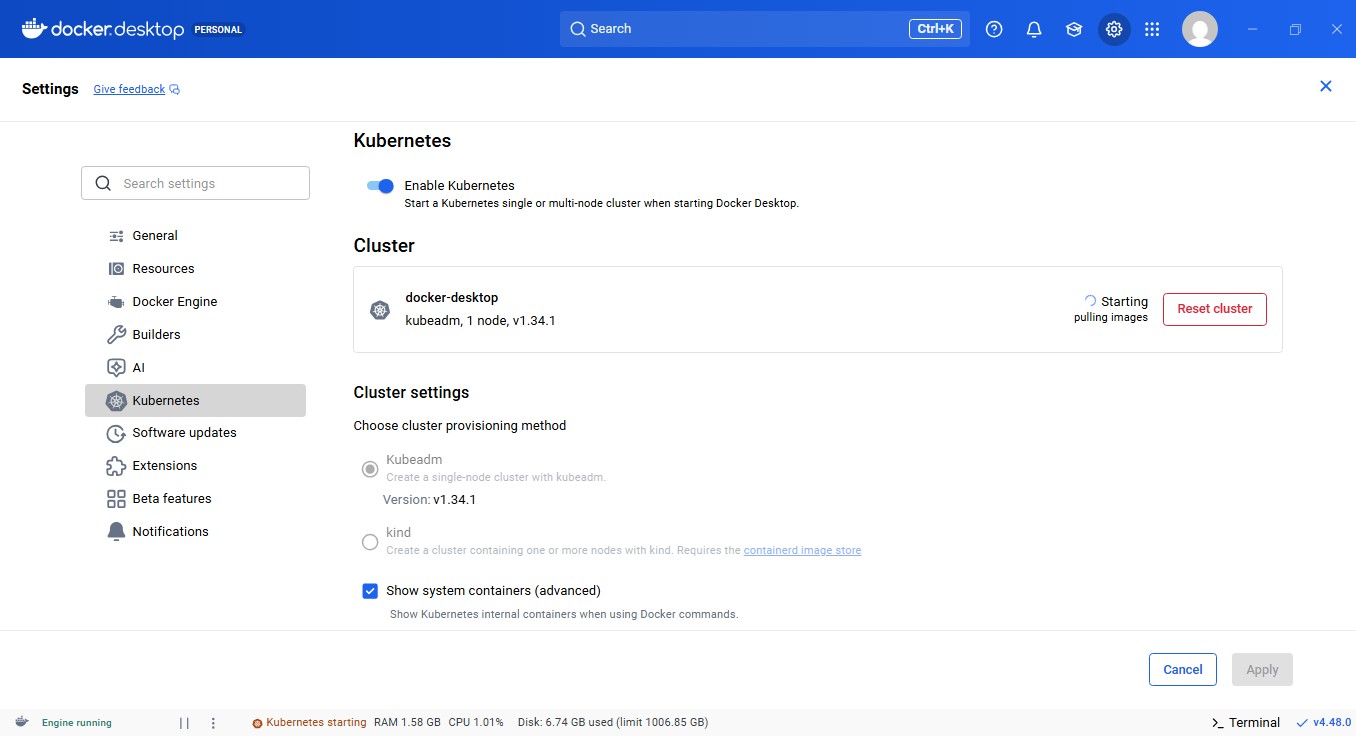
The Kubernetes server runs locally within your Docker instance, is not configurable, and is a single-node cluster. It runs within a Docker container on your local system, and is only for local testing.

Turning on Kubernetes allows you to deploy your workloads in parallel, on Kubernetes, Swarm, and as standalone containers. Turning on or off the Kubernetes server does not affect your other workloads.

# Lab Procedure

[**Install and turn on Kubernetes**](https://docs.docker.com/desktop/features/kubernetes/#install-and-turn-on-kubernetes)

1. Open the Docker Desktop Dashboard and navigate to Settings.
2. Select the Kubernetes tab.
3. Toggle on Enable Kubernetes.
4. Choose your [cluster provisioning method.](https://docs.docker.com/desktop/features/kubernetes/#cluster-provisioning-method)
5. Select Apply to save the settings.



This sets up the images required to run the Kubernetes server as containers, and installs the kubectl command-line tool on your system at:

/usr/local/bin/kubectl (Mac) or

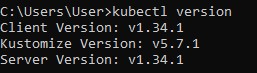
C:\Program Files\Docker\Docker\resources\bin\kubectl.exe (Windows).

Docker Desktop for Linux does not include kubectl by default. You can install it separately by following theKubernetes installation guid[e.](https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/) Ensure the kubectl binary is installed at /usr/local/bin/kubectl.

When Kubernetes is enabled, its status is displayed in the Docker Desktop Dashboard footer and the Docker menu.

Docker Desktop does not upgrade your Kubernetes cluster automatically after a new update. To upgrade your Kubernetes cluster to the latest version, select **Reset Kubernetes Cluster**

You can check which version of Kubernetes you're on with:  **kubectl version**



[**Using the kubectl command**](https://docs.docker.com/desktop/features/kubernetes/#using-the-kubectl-command)

Kubernetes integration automatically installs the Kubernetes CLI command at: **/usr/local/bin/kubectl** on Mac and

**C:\Program Files\Docker\Docker\Resources\bin\kubectl.exe** on Windows.

This location may not be in your shell's **PATH** variable, so you may need to type the full path of the command or add it to the **PATH**.

If you have already installed **kubectl** and it is pointing to some other environment, such as **minikube** or a Google Kubernetes Engine cluster, ensure you change the context so that **kubectl** is pointing to **docker-desktop**:

**kubectl config get-contexts kubectl config use-context docker-desktop**



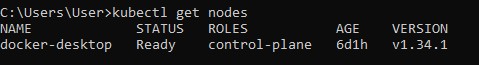
**Tip**

If the kubectl config get-contexts command returns an empty result, try:

* Running the command in the Command Prompt or PowerShell.
* Setting the KUBECONFIG environment variable to point to your .kube/config file.

[**Verify installation**](https://docs.docker.com/desktop/features/kubernetes/#verify-installation)

To confirm that Kubernetes is running, list the available nodes: kubectl get nodes



If you installed kubectl using Homebrew, or by some other method, and experience conflicts, remove /usr/local/bin/kubectl.

**EXPERIMENT-09**

**AIM: Automate the process of running containerized application developed in exercise 7 using Kubernetes**

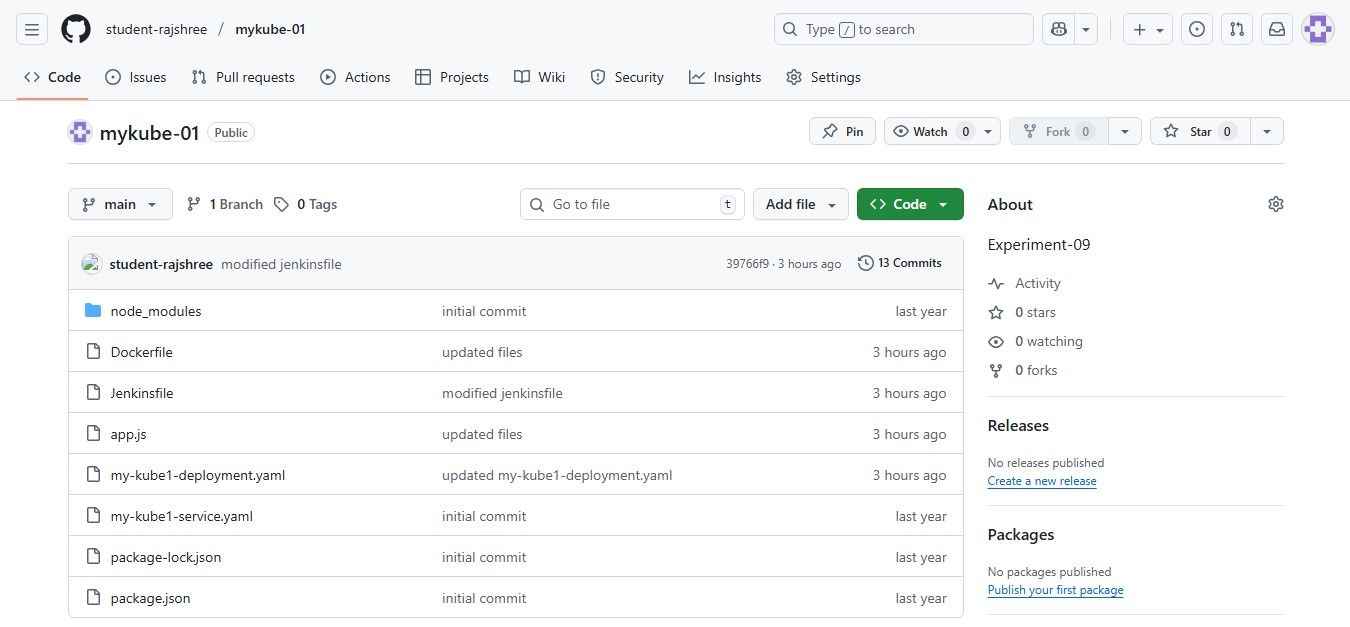
**Objective:** To automate the process of building, running, and managing containerized applications using Docker and scripting techniques.

# Software Requirements

* Docker installed ([https://www.docker.com/products/docker-desktop/)](https://www.docker.com/products/docker-desktop/)
* Kubernetes installed (Minikube or Docker Desktop with Kubernetes enabled)
* kubectl command-line tool
* Code editor (VS Code or any)
* Sample containerized application

**Step1:**

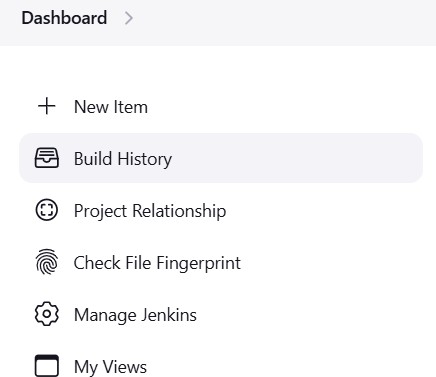
1. Clone this repository to your local repository <https://github.com/shiv4j/kube>
2. Make sure that cloned repository consist of “my-kube1deployment.yaml”, “my-kube1-service.yaml” files in folder.
3. Push this local repository to github (or) you can fork that repository from <https://github.com/shiv4j/kube>
4. After completion of pushing or forking of kube1 folder into your github repository you should able to see like this that contains all the files.



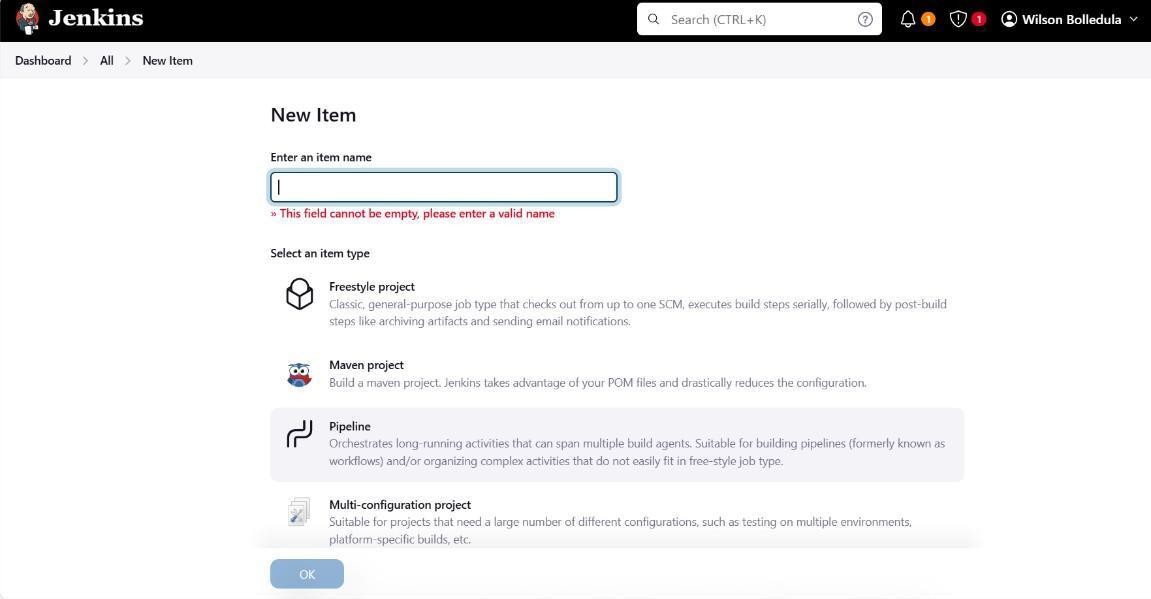
**Step2:**

Push your github repository to the Jenkins.

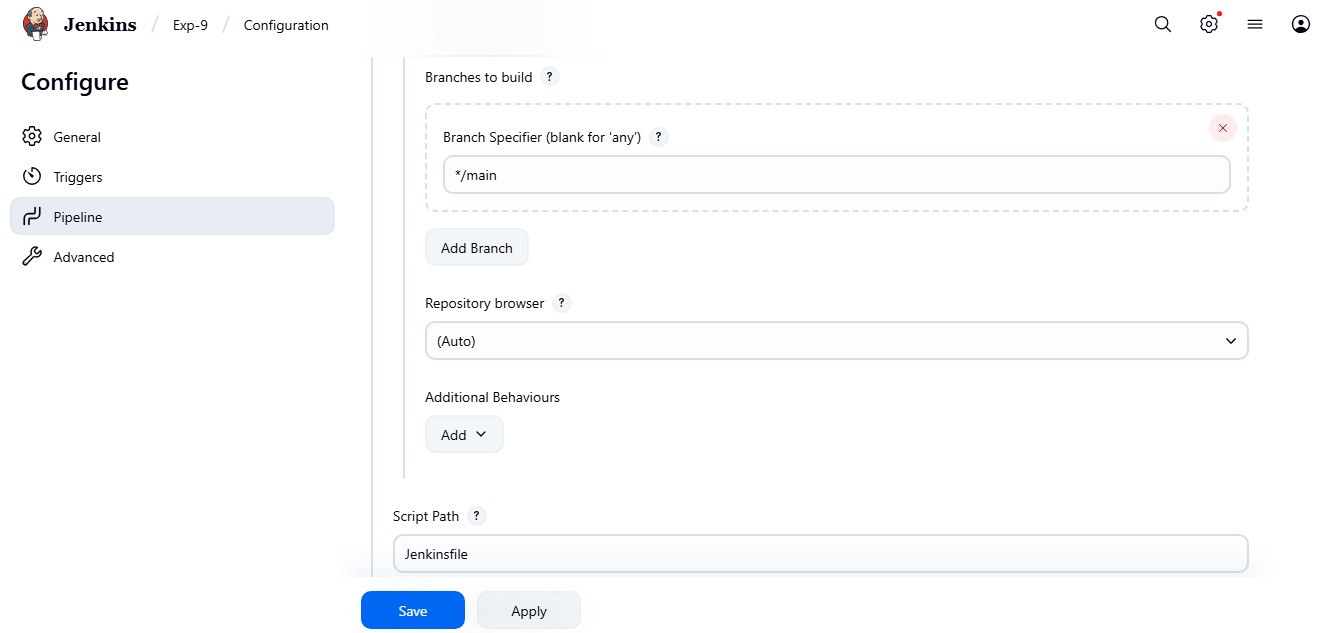
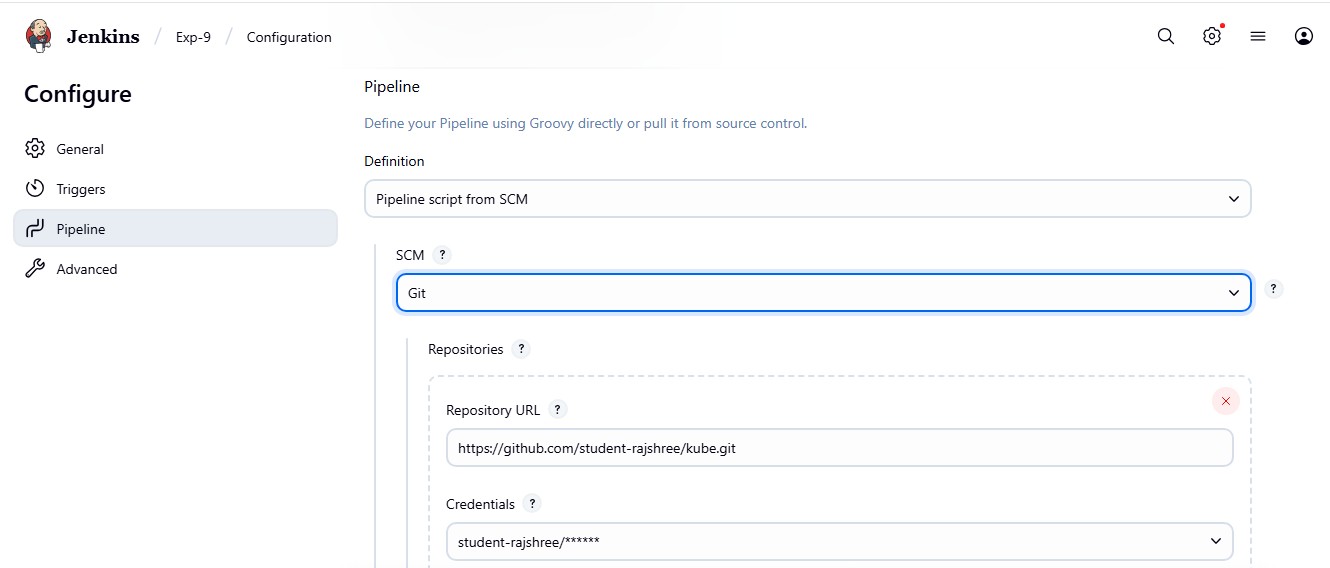
Click on New Item



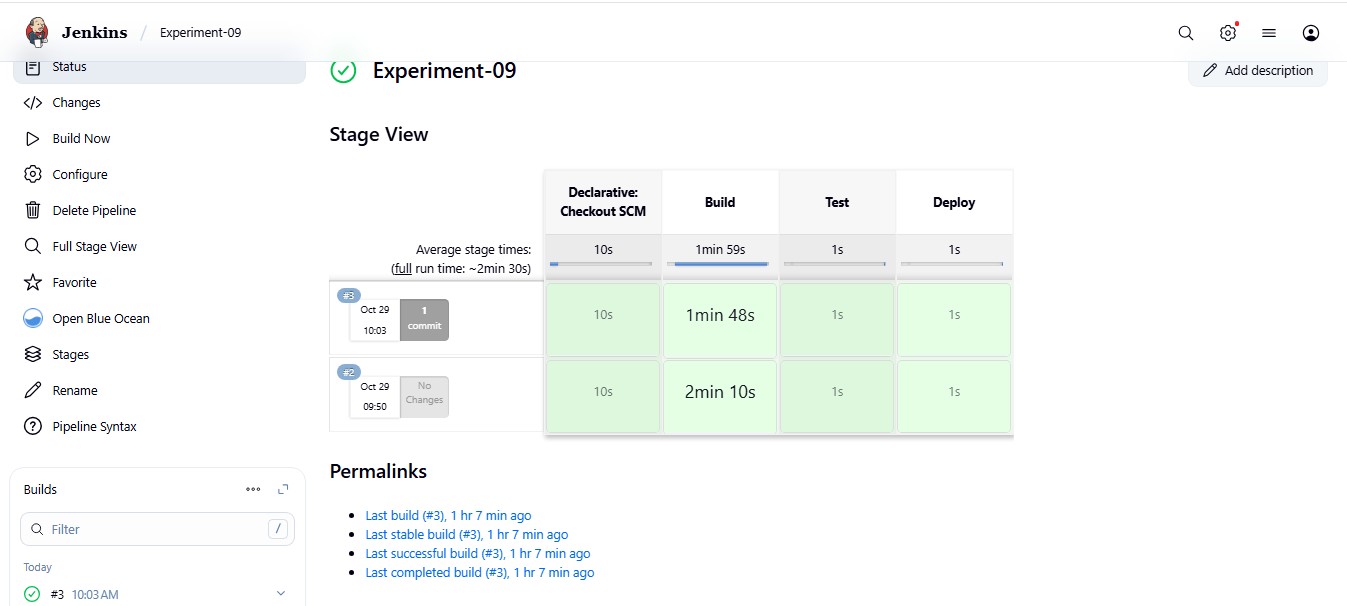
Enter a name and select Pipeline project and click on Ok.



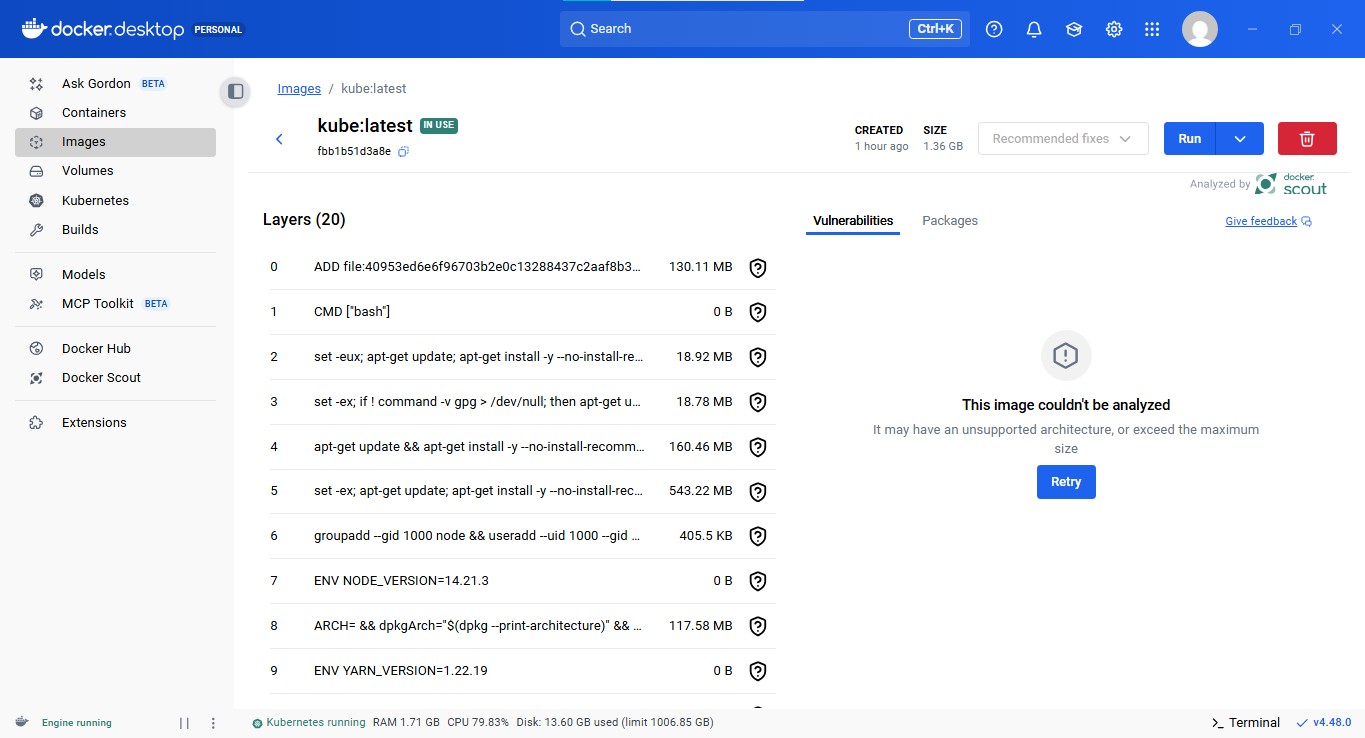
* In the configure, go to pipeline tab-> Select Definition as **Pipeline script from SCM-> Select SCM as Git** and paste your repository url
* Specify your branch whether it is main or master based on your github repository.
* Click on Apply and Save.



* After creation of your Jenkins project build it-> Click on **Build Now** The build should be shown in green tick mark.



You will get docker image for this project, like showed in the below



**Step3:**

* Push the docker image into dockerhub
* open command prompt and run the command “docker login”
* Tag your iamge using this syntax

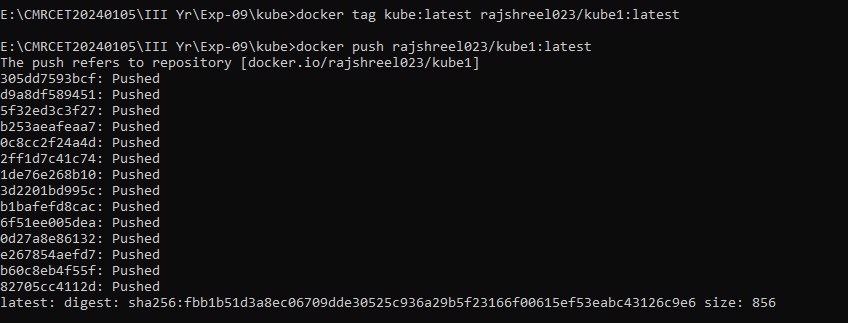
**docker tag <local-image-name>:<tag> yourusername/image-name:<tag>**

**Ex:** docker tag kube:latest rajshreel023/kube1:latest

Here username is your dockerhub account username

* Push the image to dockerhub **docker push yourusername/image-name:<tag>**

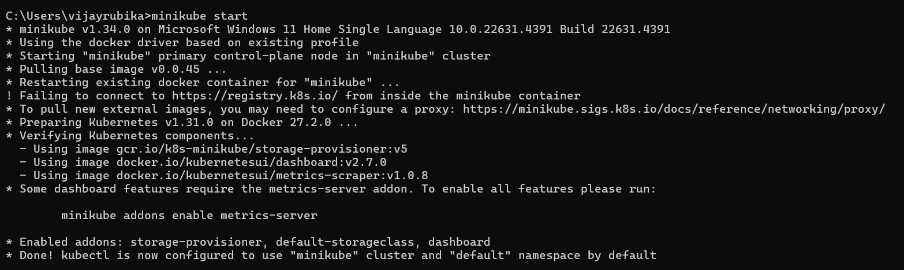
**Ex: docker push rajshreel023/kube1:latest**



**Step4:**

Start the Kubernetes

Syntax: minikube start



* Apply my-kube1-deployment.yaml file **kubectl apply -f my-kube1-deployment.yaml**



* Apply my-kube1-service.yaml file

Syntax: **kubectl apply -f my-kube1-service.yaml**



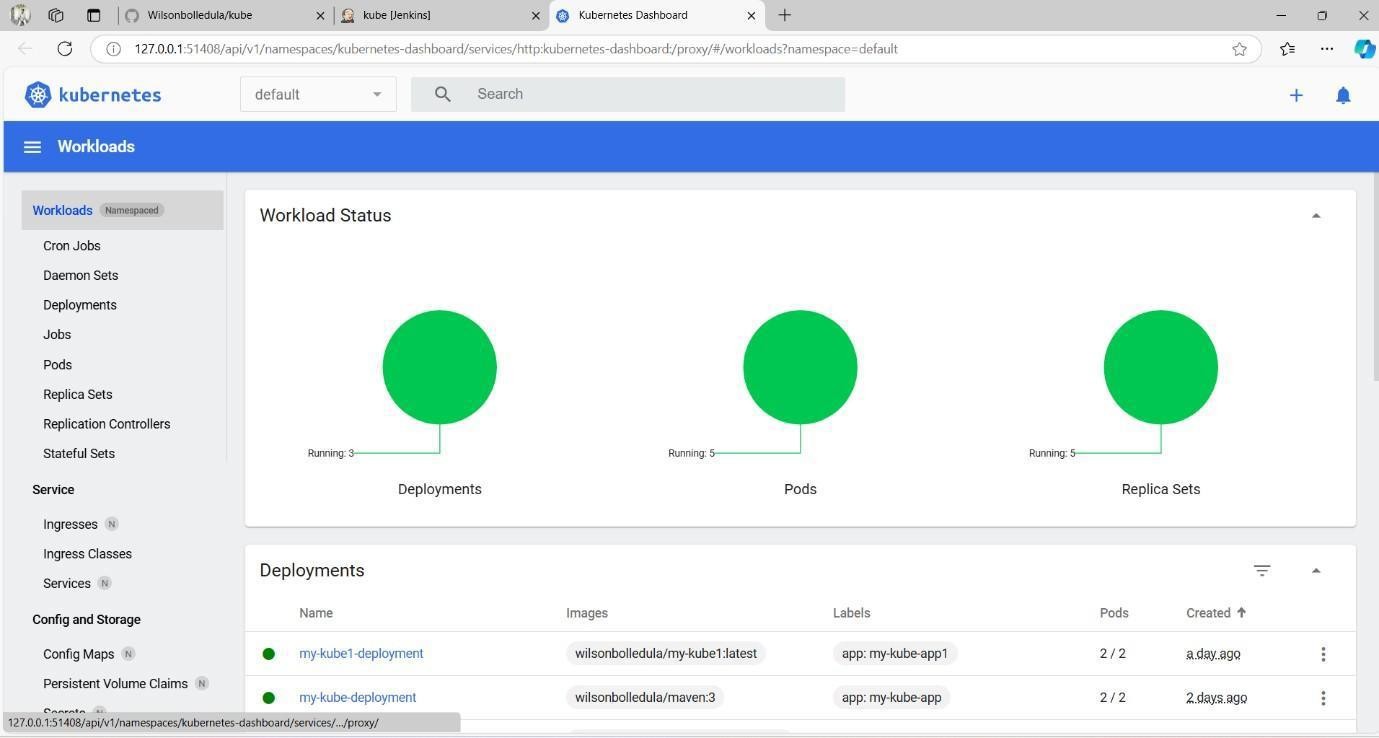
That will apply to the deployments and services

* To check that type command “**kubectl get pods**” and “**kubectl get service**

”

* To open the Kubernetes dashboard type = “minikube dashboard”

That will open Kubernetes dashboard automatically on your default primary browser



* Finally you can checkout your deployments and pods here.

**EXPERIMENT-10**

**AIM: Install and Explore Selenium for Automated Testing**

**Objective:** To install Selenium and explore its environment for automating web application testing.

# Software Requirements

* Python installed ([https://www.python.org/downloads/)](https://www.python.org/downloads/)
* Selenium library installed
* Web browser (Chrome, Firefox, etc.)
* WebDriver for the browser (e.g., ChromeDriver)
* Code editor (VS Code or any)

# Lab Procedure

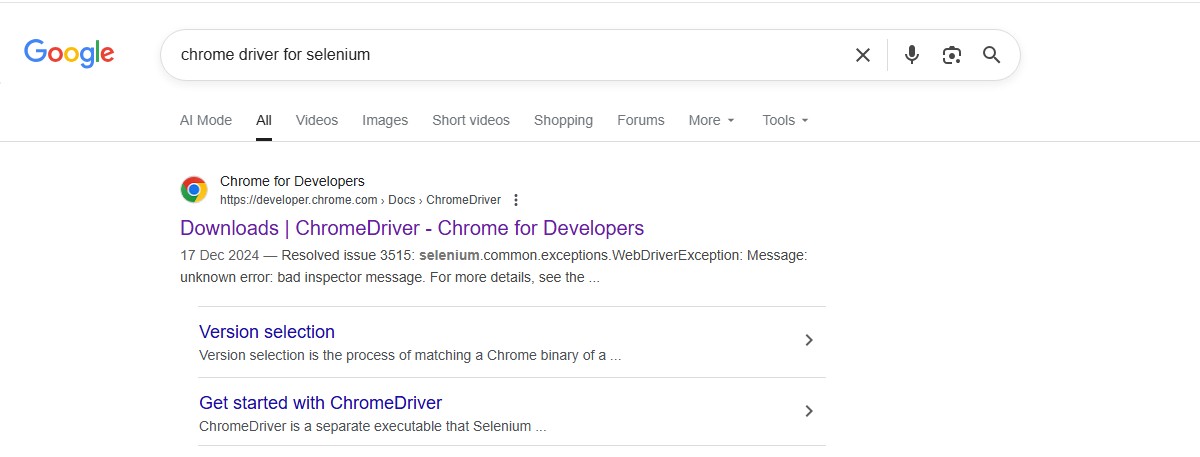
1. **Install Python** 
   * Download and install Python.
   * Verify installation:

python –version

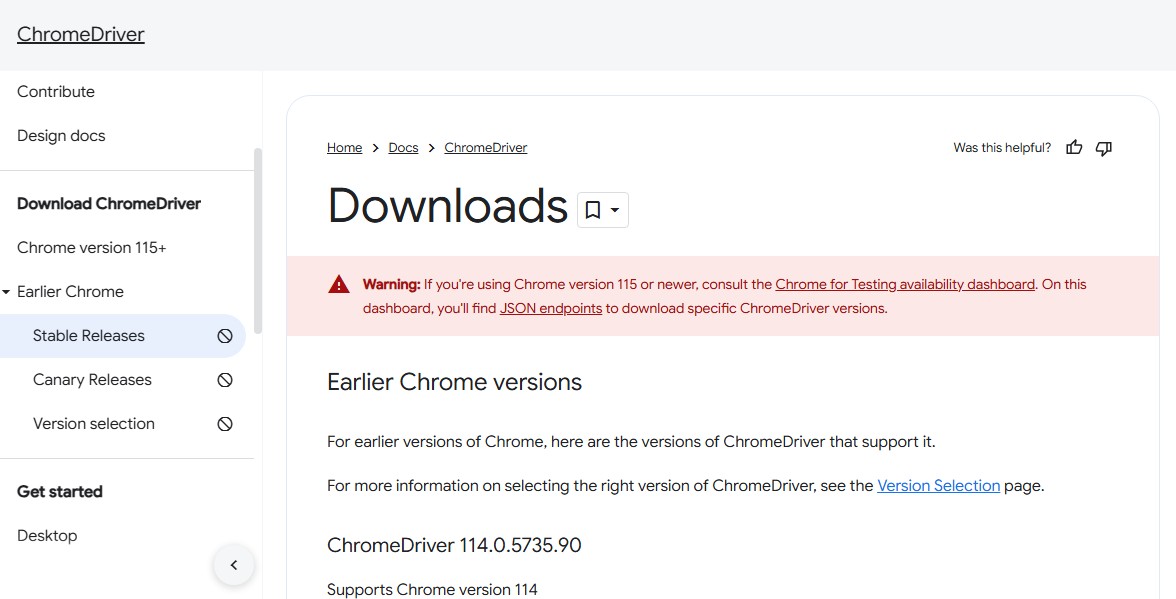
1. **Install Selenium Library** 
   * Use pip to install Selenium:

pip install selenium

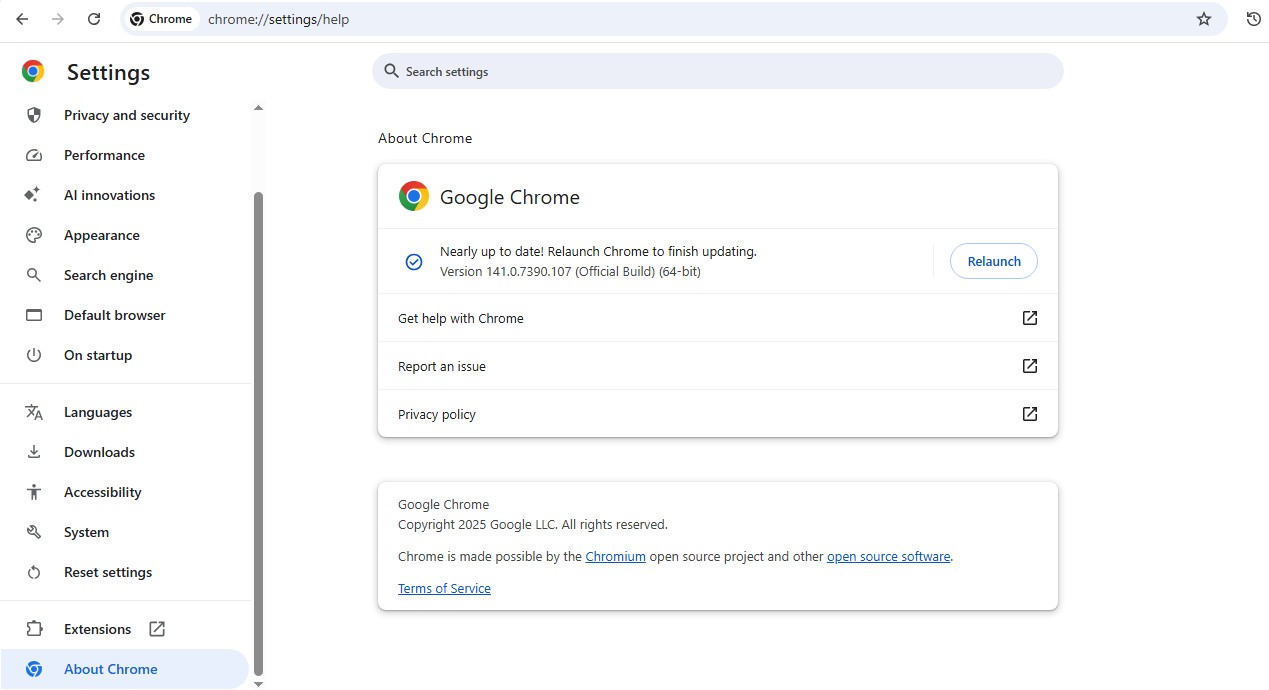
1. **Download WebDriver** 
   * Download ChromeDriver from
   * [(https://sites.google.com/chromium.org/driver/)](https://sites.google.com/chromium.org/driver/) or appropriate WebDriver for your browser.
   * Place the WebDriver executable in your system PATH or project directory.



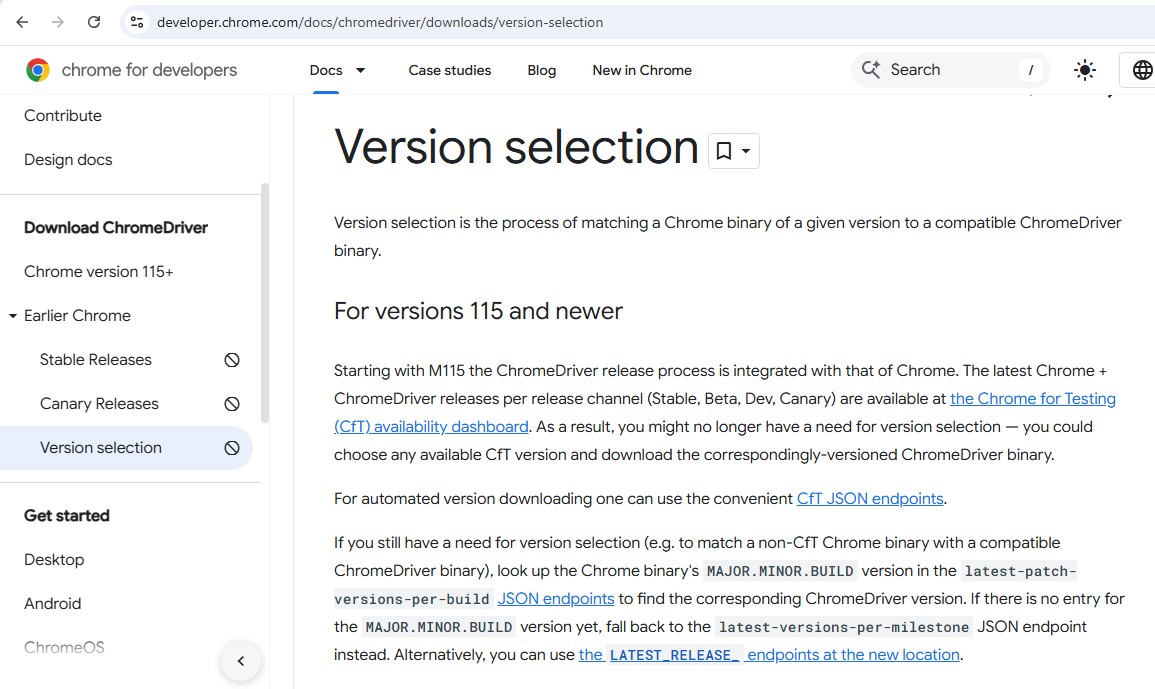
Open the 1st link.



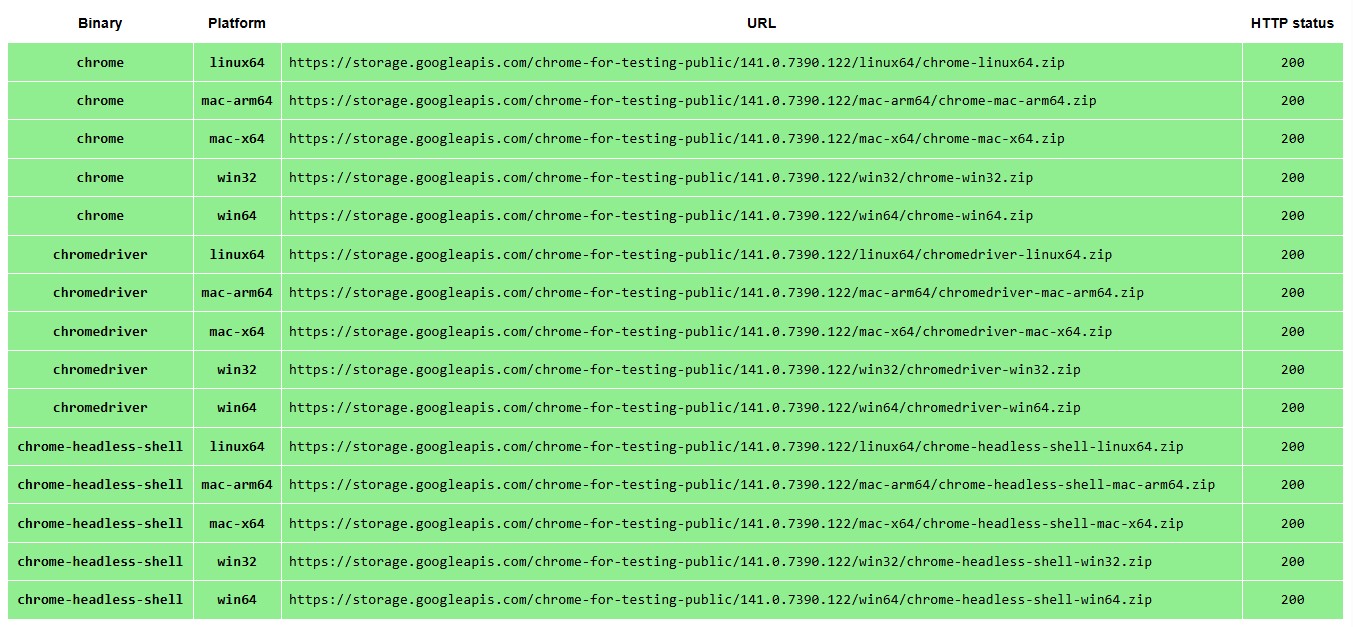
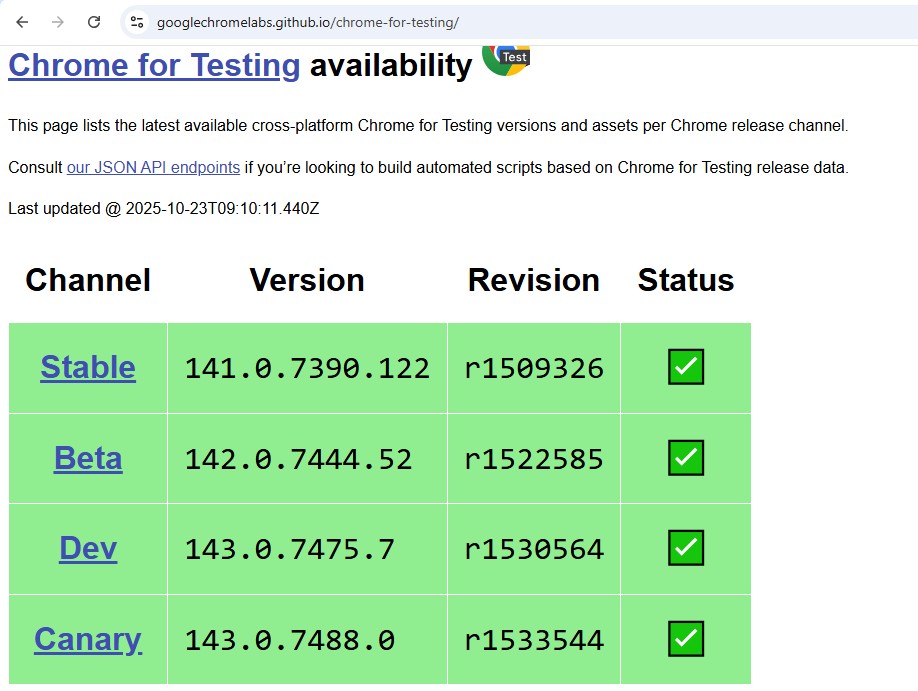
Click on version selection and check your version in chrome by chrome://settings/help



Click on **the Chrome for Testing (CfT) availability dashboard.**



Click on **Stable**



* + Select according to you Operating System and Copy and paste chromedriver link in chrome it will download automatically.

* + The file will be downloaded, extract the files and set the path of chromedriver.exe in system environment variables

* + Verify chromeDriver installation **chromeDriver –version**

* + Open a VS code and create a python file and Add the following code

from selenium import webdriver from selenium.webdriver.chrome.service import Service from selenium.webdriver.common.by import By from selenium.webdriver.common.keys import Keys import time

# Specify the correct path to your ChromeDriver chrome\_service = Service("C:\Program Files\chromedriverwin64\chromedriver.exe")

# Set up the WebDriver for Chrome

driver = webdriver.Chrome(service=chrome\_service)

try:

# Step 1: Open Google in the browser

driver.get("https://www.google.com")

# Step 2: Locate the search box using the name attribute search\_box = driver.find\_element(By.NAME, "q")

# Step 3: Enter the search term and press Enter search\_box.send\_keys("https://cmrcet.ac.in/")

search\_box.send\_keys(Keys.RETURN)

# Optional: Wait for a few seconds to see the search results time.sleep(20)

finally:

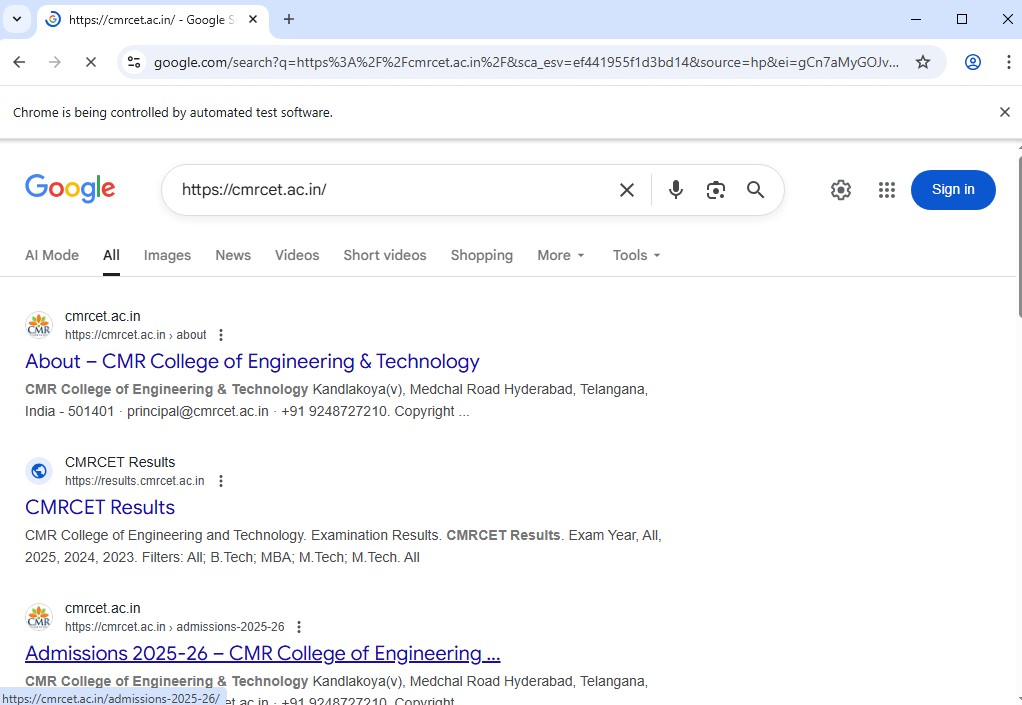
# Step 4: Close the browser

driver.quit()

**Make sure to change the path of the driver in the code to avoid errors**

 Run the code

**OUTPUT**



**EXPERIMENT-11**

**AIM:** Write a simple program in JavaScript and perform testing using Selenium.

**Objective:** To develop automated test cases using Selenium for validating the functionality of a previously containerized web application.

# Software Requirements

* Python or Node.js environment (as per project)
* Selenium WebDriver installed
* Web browser (Chrome, Firefox, etc.)
* WebDriver for the browser (e.g., ChromeDriver)
* Docker installed and running
* Code editor (VS Code or any)
* Sample containerized web application running

# Lab Procedure

1. Open VS code and create html and javascript files 2. Create html file and paste this code **index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Sum Calculator</title>

<style> body{

text-align: center;

}

</style>

</head>

<body>

<h1>Sum Calculator</h1>

<input type="number" id="num1" placeholder="Enter first number">

<input type="number" id="num2" placeholder="Enter second number">

<button id="add">Add</button>

<p>Result: <span id="result">0</span></p>

<script> function calculateSum(a, b)

{ return a + b;

} document.getElementById('add').addEventListener('click', function() { const num1

= parseInt(document.getElementById('num1').value, 10); const num2 = parseInt(document.getElementById('num2').value, 10); const result = calculateSum(num1, num2); document.getElementById('result').textContent = result; });

</script>

</body>

</html>

**script.js** import { fileURLToPath } from 'url'; import { dirname } from 'path'; import { Builder, By, until } from 'selenium-webdriver'; import assert from 'assert';

// Define \_\_dirname manually const \_\_filename = fileURLToPath(import.meta.url); const \_\_dirname = dirname(\_\_filename);

async function runTest() { // Define driver here let driver = await new Builder().forBrowser('chrome').build();

try {

// Use \_\_dirname (not dirname) await driver.get('file://' + \_\_dirname + '/index.html');

// Find input elements const num1 = await driver.findElement(By.id('num1')); await num1.sendKeys('50');

const num2 = await driver.findElement(By.id('num2')); await num2.sendKeys('10');

// Click the "Add" button const addButton = await driver.findElement(By.id('add')); await addButton.click();

// Verify the result const result = await driver.findElement(By.id('result')); const text = await result.getText();

assert.strictEqual(text, '60', 'Sum calculation is incorrect'); console.log('✅ Test passed: Sum is correct');

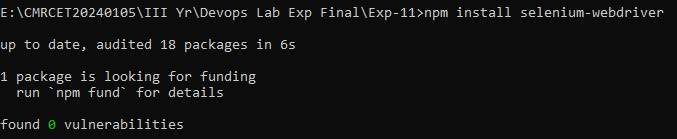
} catch (error) { console.error('❌ Test failed:', error);

} finally { console.log('Press any key to exit...'); process.stdin.setRawMode(true); process.stdin.resume(); process.stdin.on('data', async () => { await driver.quit(); process.exit(0);

});

} } runTest();

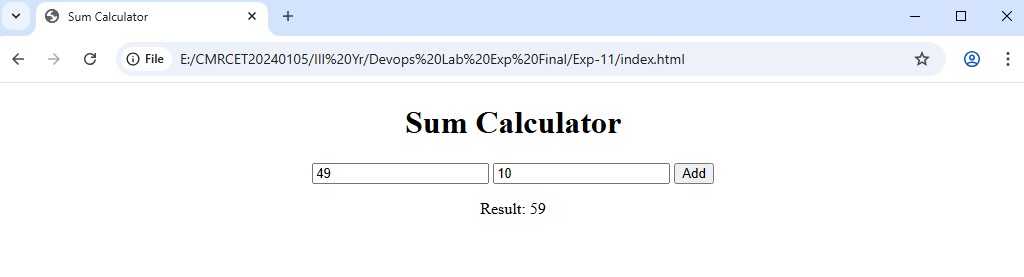
1. Open terminal
2. npm install selenium-webdriver



1. node script.js



**OUTPUT**



**EXPERIMENT-12**

**AIM:** Develop test cases for the above containerized application using selenium.

**Objective:** To validate the functional, UI, authentication, and integration behavior of the web application running inside Docker by automating end-to-end tests using Selenium (remote WebDriver / Selenium container).

# Software Requirements

* Code editor (VS Code or any)
* Docker Desktop
* Python (for running scripts locally if needed)
* Selenium (pip install selenium)
* Web browser (Chrome, Firefox, etc.)

# Lab Procedure

Clone this repository to your local repository <https://github.com/Srivaishnavi08/tests>

**Or follow the below steps Step-1:**

Create a directory named selenium-test and navigate to the current directory path. tests/

**├──** Dockerfile ├── index.html

├── SeleniumTest.py

├─ docker-compose.yml

**Step 2:** Create the index.html file

This is your sample web page that Selenium will interact with.

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Selenium Login Example</title>

</head>

<body>

<!-- Homepage -->

<div id="homepage">

<h1>Welcome!!!!!!</h1>

<!-- Get Started Free Button -->

<a href="#loginPage" id="get-started" onclick="navigateToLogin()">Get started</a>

</div>

<!-- Login Page -->

<div id="loginPage" style="display: none;">

<h2>Login to the Page</h2>

<form onsubmit="return validateLogin()">

<label for="user\_email\_login">Email:</label>

<input type="email" id="user\_email\_login" name="user\_email\_login" required>

<br><br>

<label for="user\_password">Password:</label>

<input type="password" id="user\_password" name="user\_password" required>

<br><br>

<button type="submit" name="commit">Login</button>

</form>

<p id="error-message" style="color: red; display: none;">Invalid credentials, please try again.</p>

</div>

<!-- Dashboard Section (only shown after successful login) -->

<div id="dashboard" style="display: none;">

<h2>Welcome to Your Dashboard!</h2>

<p>This is the dashboard area you see after a successful login.</p> </div>

<script>

// Function to navigate to the login page function navigateToLogin() { document.getElementById('homepage').style.display = 'none'; document.getElementById('loginPage').style.display = 'block';

}

// Function to validate login credentials function validateLogin() { const email = document.getElementById('user\_email\_login').value; const password = document.getElementById('user\_password').value;

if (email === "abc@gmail.com" && password === "password") {

// Hide login page and display dashboard document.getElementById('loginPage').style.display = 'none'; document.getElementById('dashboard').style.display = 'block'; return false; // Prevent actual form submission

} else {

// Show error message if credentials are incorrect document.getElementById('error-message').style.display = 'block'; return false; // Prevent actual form submission

}

}

</script>

</body>

</html>

**Step 3: Create the dockerfile**

This dockerfile sets up a simple HTTP server to serve your index.html file.

**Dockerfile**

# Use the official Python image as the base image

FROM python:3.9

# Set the working directory in the container

WORKDIR /app

# Copy the index.html file to the container COPY index.html .

# Expose port 8000 for the HTTP server

EXPOSE 8000

# Start a simple HTTP server to serve the index.html file

CMD ["python", "-m", "http.server", "8000"]

**Step-4: Create the Selenium Test Script (seleniumTest.py)** This script will automate testing of your HTML page using Selenium.

from selenium import webdriver from selenium.webdriver.common.by import By from selenium.webdriver.support.ui import WebDriverWait from selenium.webdriver.support import expected\_conditions as EC import time

print("Test Execution Started") options = webdriver.ChromeOptions() options.add\_argument('--ignore-ssl-errors=yes') options.add\_argument('--ignore-certificate-errors')

# Start the Selenium WebDriver driver = webdriver.Remote( command\_executor='http://localhost:4444/wd/hub', options=options

)

# Maximize the window size driver.maximize\_window() time.sleep(10) driver.get("http://host.docker.internal:8000") # Access the local server time.sleep(10)

try:

# Wait for the "Get started free" link to be clickable link = WebDriverWait(driver, 30).until(

EC.element\_to\_be\_clickable((By.LINK\_TEXT, "Get started"))

)

link.click() # Click the link time.sleep(10) # Wait for any resulting page to load

WebDriverWait(driver, 10).until(

EC.presence\_of\_element\_located((By.ID, "user\_email\_login"))

)

WebDriverWait(driver, 10).until( EC.presence\_of\_element\_located((By.ID, "user\_password"))

)

# Enter login credentials username = driver.find\_element(By.ID, "user\_email\_login") password = driver.find\_element(By.ID, "user\_password") login\_button = driver.find\_element(By.NAME, "commit")

username.send\_keys("abc@gmail.com") # Replace with actual username password.send\_keys("password") # Replace with actual password login\_button.click()

# Check for a post-login element (adjust to your page's unique element for logged-in users) try:

error\_message = WebDriverWait(driver, 10).until(

EC.visibility\_of\_element\_located((By.ID, "error-message"))

)

time.sleep(10) print("Login failed: Incorrect credentials") except:

# No error message found, proceed with checking for dashboard

WebDriverWait(driver, 10).until(

EC.visibility\_of\_element\_located((By.ID, "dashboard")) # Replace with actual post-login element ID

)

print("Login Successful!")

except Exception as e:

print(f"An error occurred while trying to click the link: {e}")

finally:

# Ensure the browser quits after execution driver.quit() print("Test Execution Completed!")

**Step 5:** Create the docker-compose.yml File

This file defines two services: your HTML server and the Selenium Chrome container.

**(docker-compose.yml)** services: app: build: context: .

dockerfile: Dockerfile container\_name: html-server ports:

* "8000:8000"

selenium:

image: selenium/standalone-chrome container\_name: selenium-chrome ports:

* "4444:4444" depends\_on:
* app

**Step-6:**

* Build and Run Your Docker Containers
* In terminal use the command

**docker compose –f docker-compose up --build** This command will:

* + Build the Docker image for your HTML server.
  + Pull the Selenium standalone Chrome image.



* Start both services.
* Press **v** to navigate to docker.

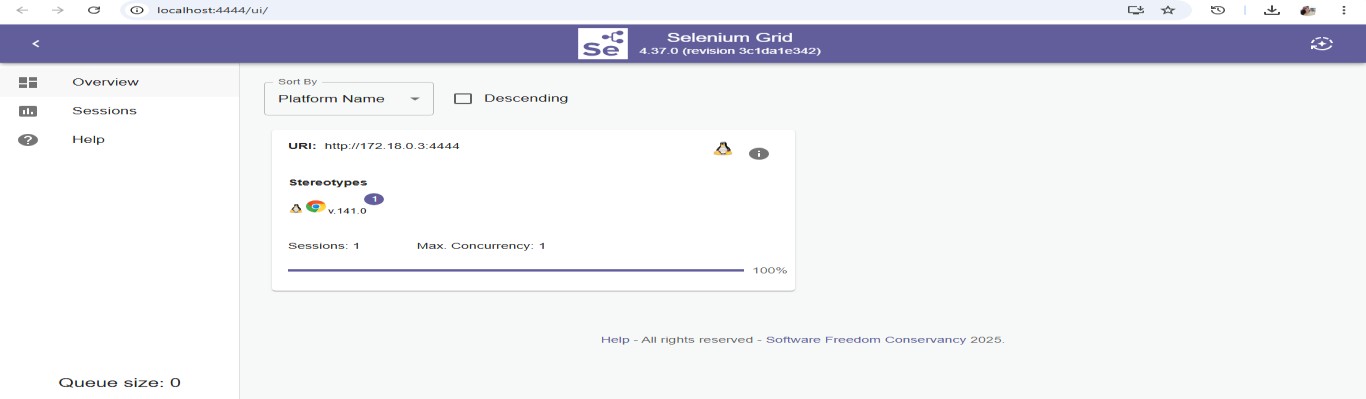


* Click on the links

**8000:8000**



**4444:4444**



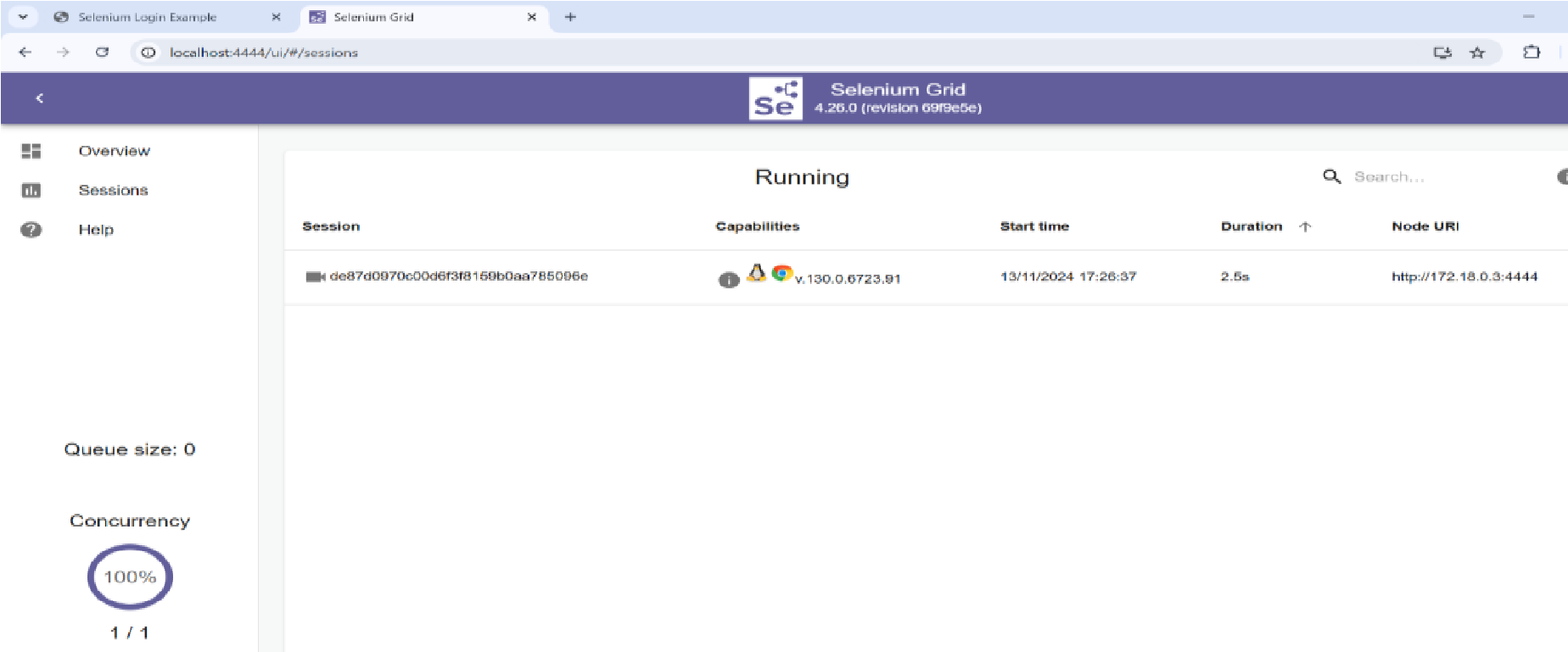
**Step 7:**

Run the Selenium Test Script

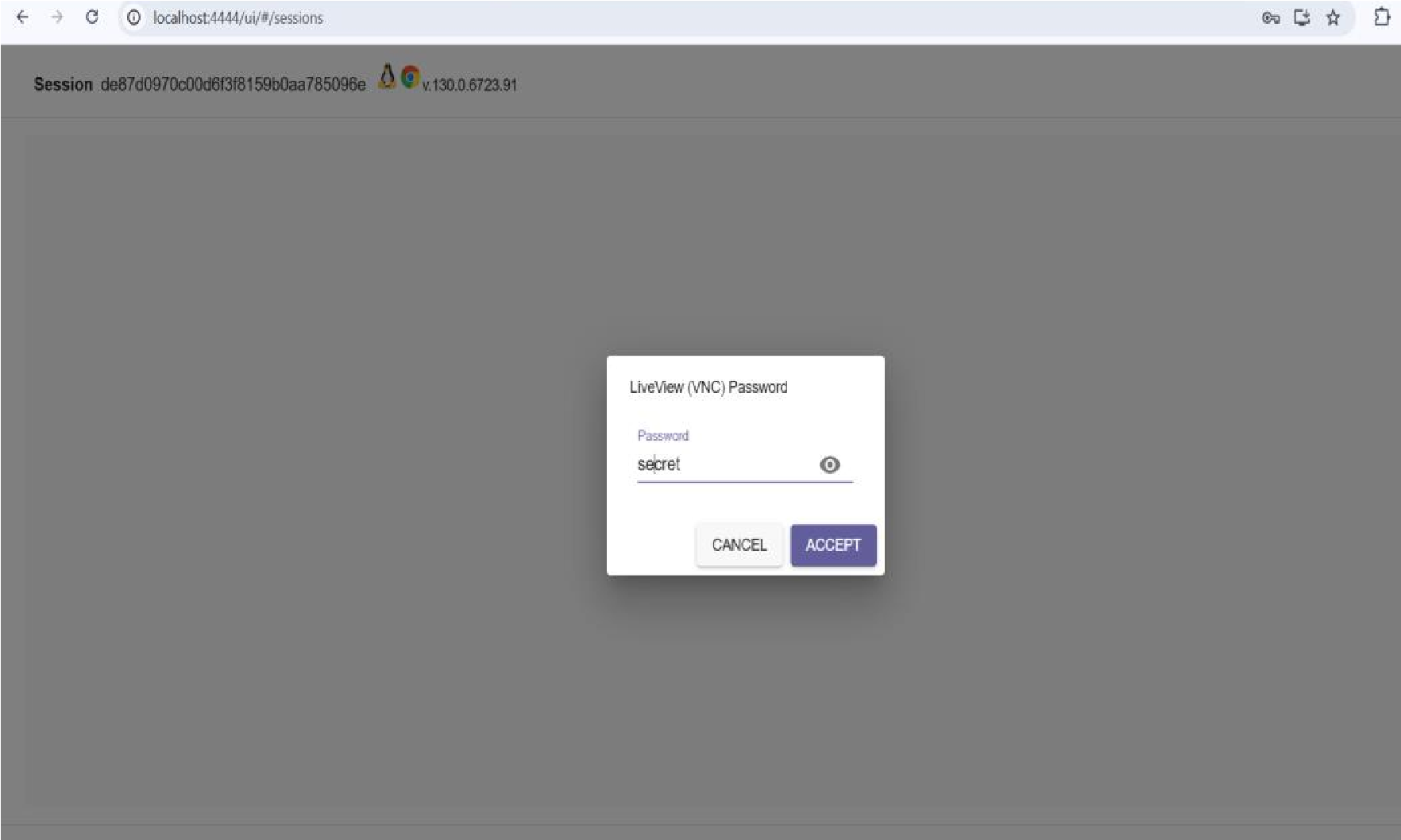
While your containers are running, open a new terminal and run:

**python seleniumTest.py**

Go to sessions, click on video icon.



Enter the password “secret” to view the video of automatic testing.



**Step 8:**

Stop the Docker Containers Once you are done testing.

Stop the containers using: **docker compose –f docker-compose down**

This command stops and removes the containers, networks, and any volumes associated with theservices.

After running the test and clicking the video icon, the video will show how the test interacts with the web page (clicking buttons, entering passwords, etc.)