1. **Write code for a simple user registration form for an event**.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>User Registration Form</title>

</head>

<body>

<form id="registrationForm">

<label for="fullName">Full Name:</label><br>

<input type="text" id="fullName" name="fullName" required><br>

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

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

<label for="password">password:</label><br>

<input type="password" id="password" name="password" required><br>

<br>

<input type="submit" value="Register">

</form>

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

<p>Registration Successful!</p>

</div>

<script>

document.getElementById("registrationForm").addEventListener("submit", function(event) {

event.preventDefault();

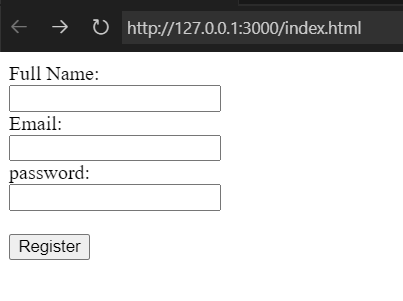
document.getElementById("successMessage").style.display = "block";

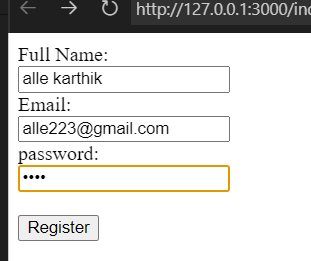
document.getElementById("registrationForm").reset();

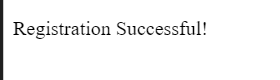
});

</script>

</body>

</html>

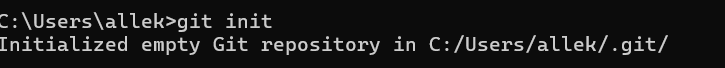




**2.Explore Git and GitHub commands**.

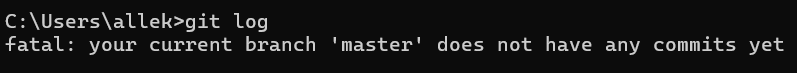
Git and GitHub are two of the most popular tools used for version control and collaboration in software development.

Here are some common Git and GitHub commands.

* Initializing a git repository: $git init
* 
* Checking the status of your repository: $ git status
* 
* Adding files to the stage: $ git add

• Committing changes: $ git commit -m "commit message"

• Checking the commit history: $ git log



• Undoing changes: $ git checkout

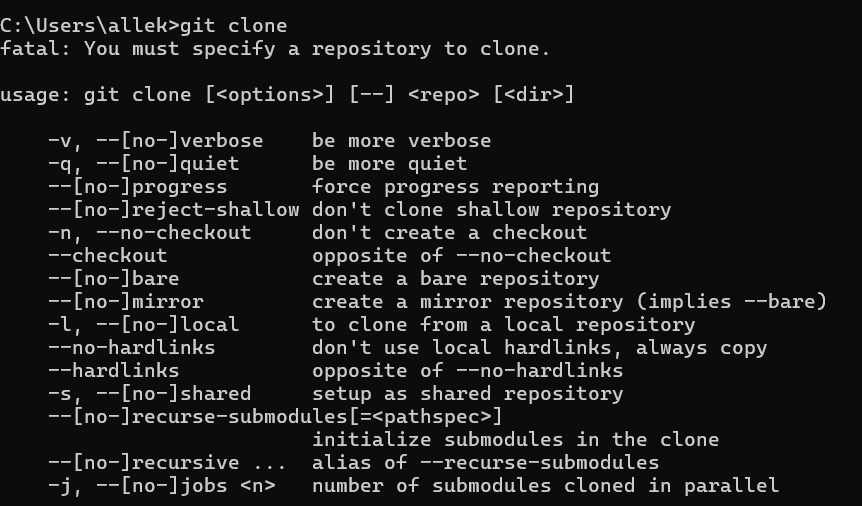
• Creating a new branch: $ git branch

• Switching to a different branch: $ git checkout

• Merging two branches: $ git merge

• Pushing changes to a remote repository: $ git push origin

• Cloning a repository from GitHub: $ git clone



• Creating a pull request on GitHub: Go to the repository on GitHub, select the branch you want to merge and click the "New pull request" button.

**3.Practice Source code management on GitHub. Experiment with the source code written in exercise 1**

Here a step by step procedure to practice source code management on github using source code written in exercise 1

1:sign up/login to github

If you don’t have a github account signup for one

If you already have an account login to github

2:create new repository

Click on “+” sign on the top right corner of the hithub page

Select new repository

Fill the repository name ,descriotion and other details

Optionally choose to initialize the repository with a readme file

Click on create repository

3:open any text editor like vs code

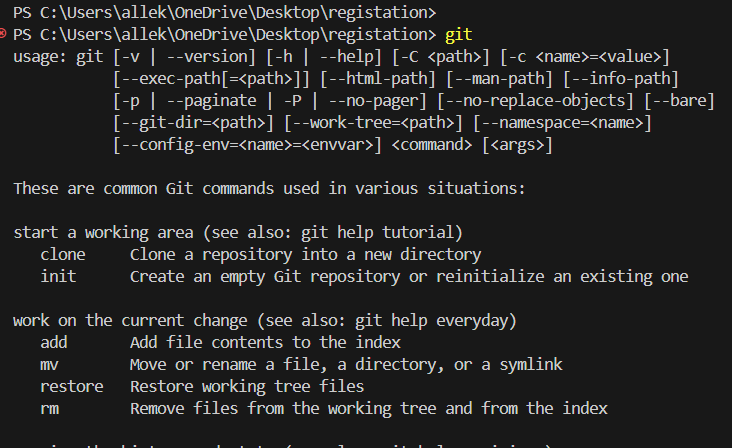
Open vs code ->terminal->check whether git command is exists or not .just type git on terminal

If git is not reconginised by vs code

->first of see whether git is installed on your pc or not if not installed then install git

->after installing the git set the path in environment variables

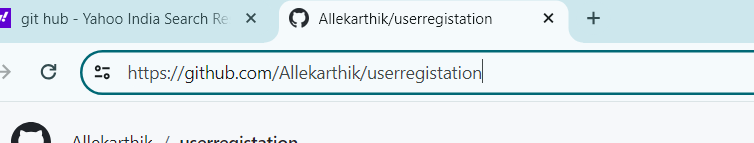
->then restart your system ,then again type git



4:whatever the files that we created for simple user registration form add them to your git repository

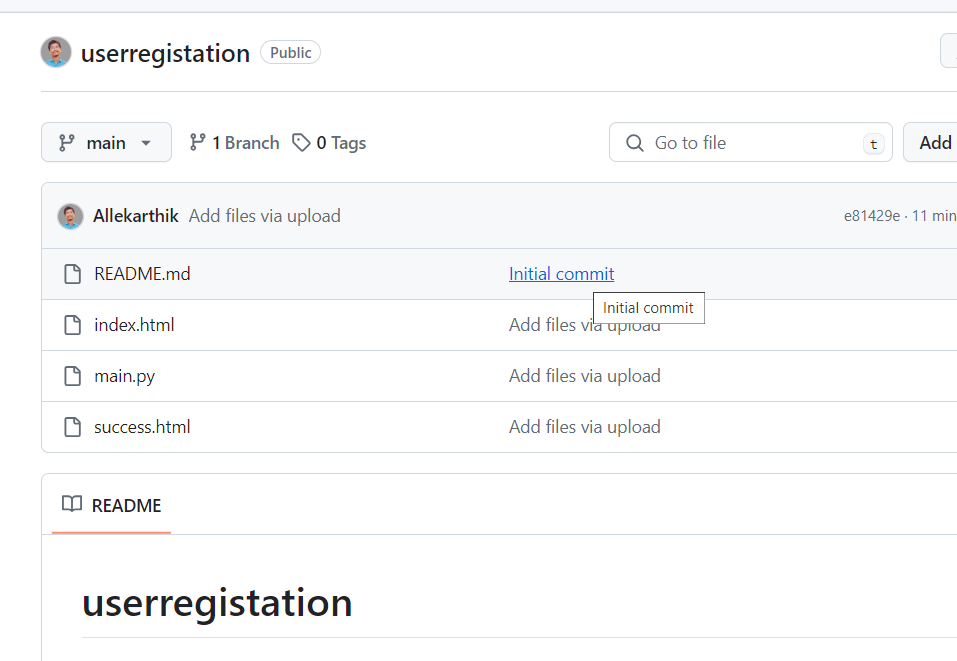
5:after adding open terminal and type git clone <https://github.com/Allekarthik/userregistation>

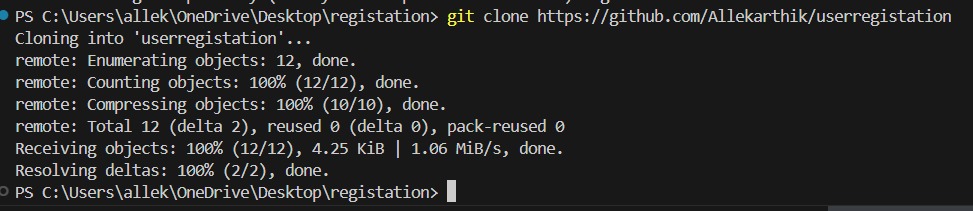
Here after git clone it should be ur link

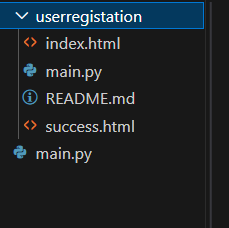


-

6:then you can access all your files in github from vs code







**4. Jenkins installation and setup, explore the environment.**

Install java jdk-21

Set environment varaiable for JDK

Download and install Jenkins

Run Jenkins on local host http://localhost:6969/

Username :admin

Password:5cfe93da2d2a444ebb1485b86c2c95ed

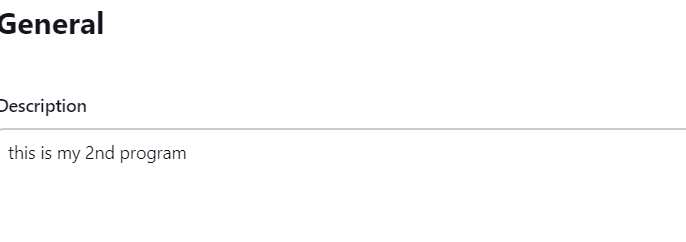
Note:it is different from user to user you have set up this after installation

Steps to run simple python program:

Go to dashboard->new item,then enter an item name choose free style project

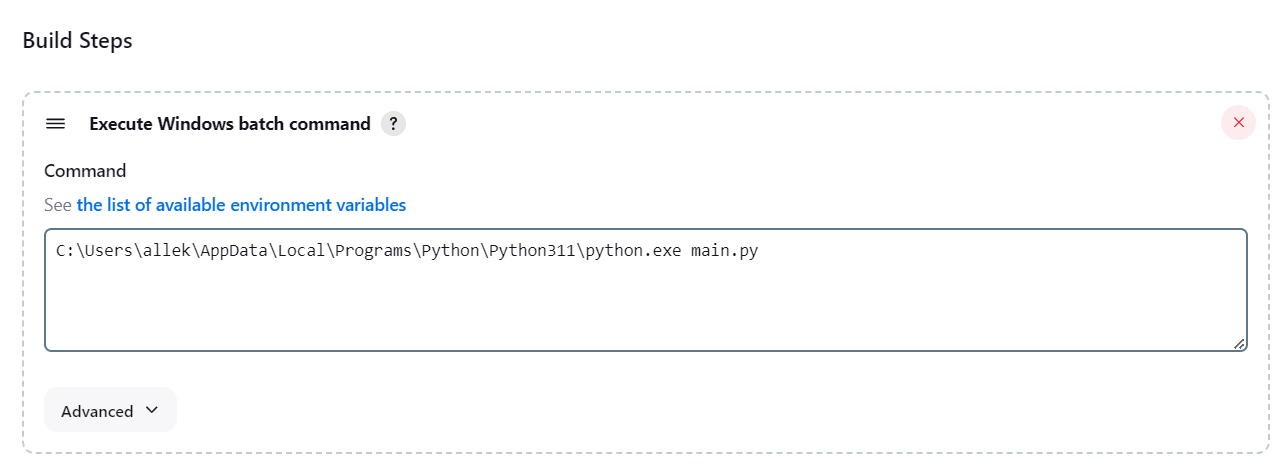
Click on ok

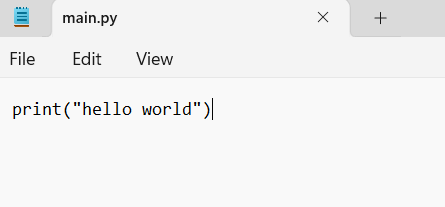
Then open general->description(eg:this is my first program)

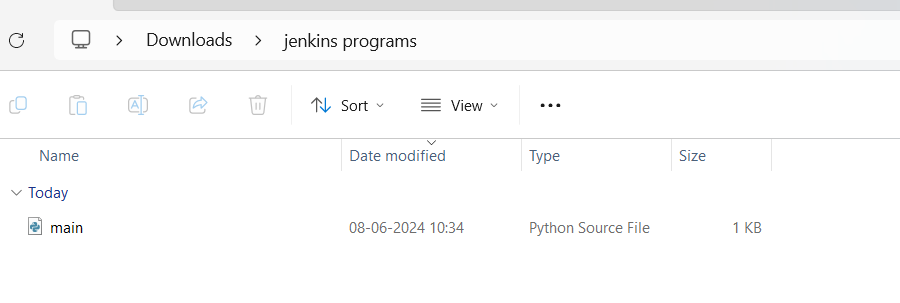


Advanced->use custom workspace->enter the path where your python file has saved eg: C:\Users\allek\Downloads\jenkins programs

Go to build steps ->execute windows batch command (you can get get this path from environment variables)



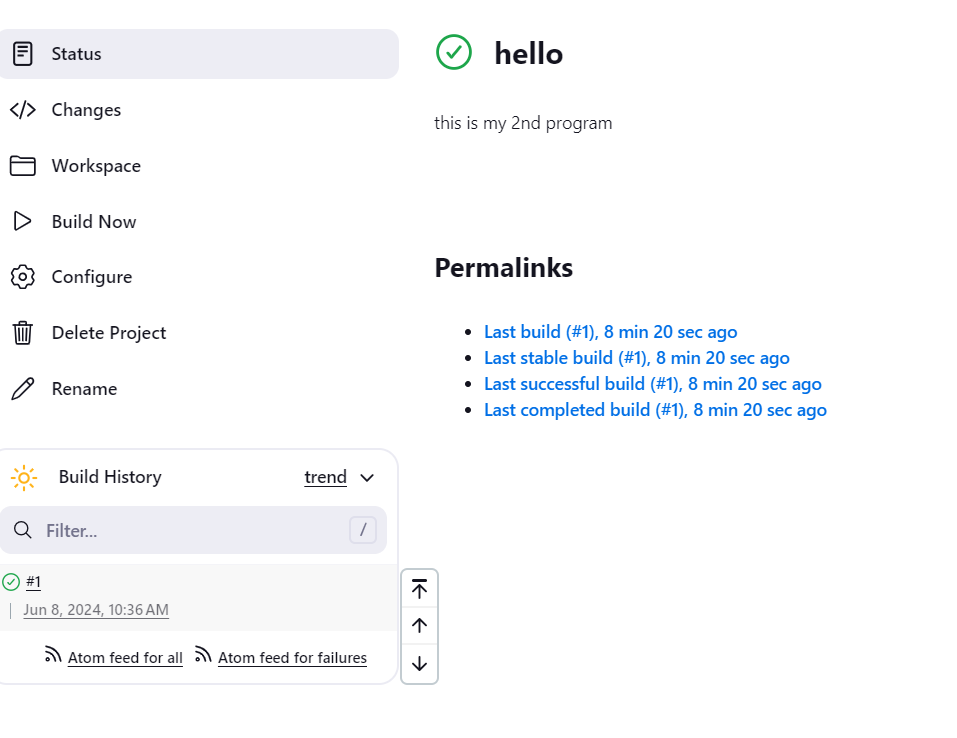




Then click on save

Go to build now,after this you can find build history open link(with date&time) and click on console output

Output will be displayed





**5.Demonstrate continuous integration and development using Jenkins.(build pipeline)**

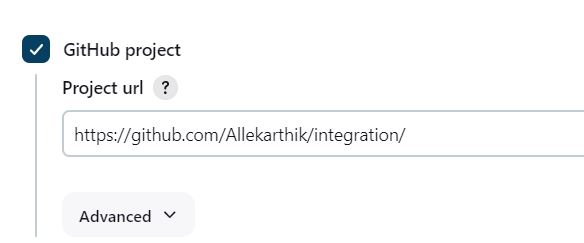
Project url :github.com/Allekarthik/integration

Repository url :https://github.com/Allekarthik/integration.git

For integration of Jenkins with github first of all we need to create a repository in that place any file lets say eg:index.html

Then create new item ->general ->add any description

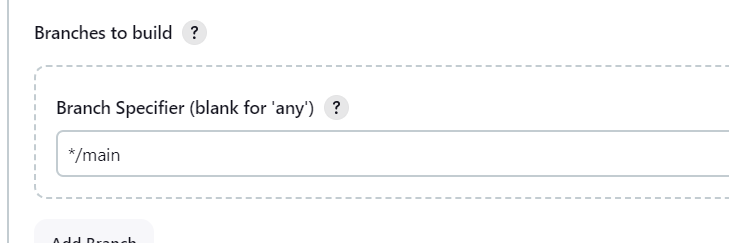
In github project add project url



Then in git add repository url

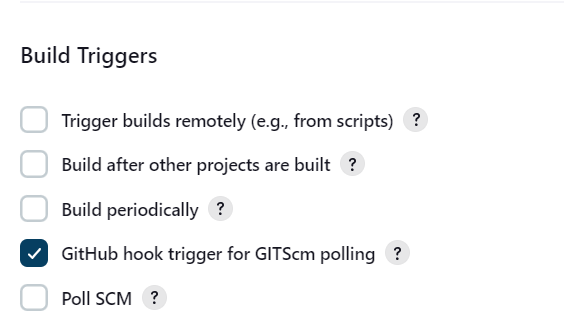


Then select main because my github is stored under main



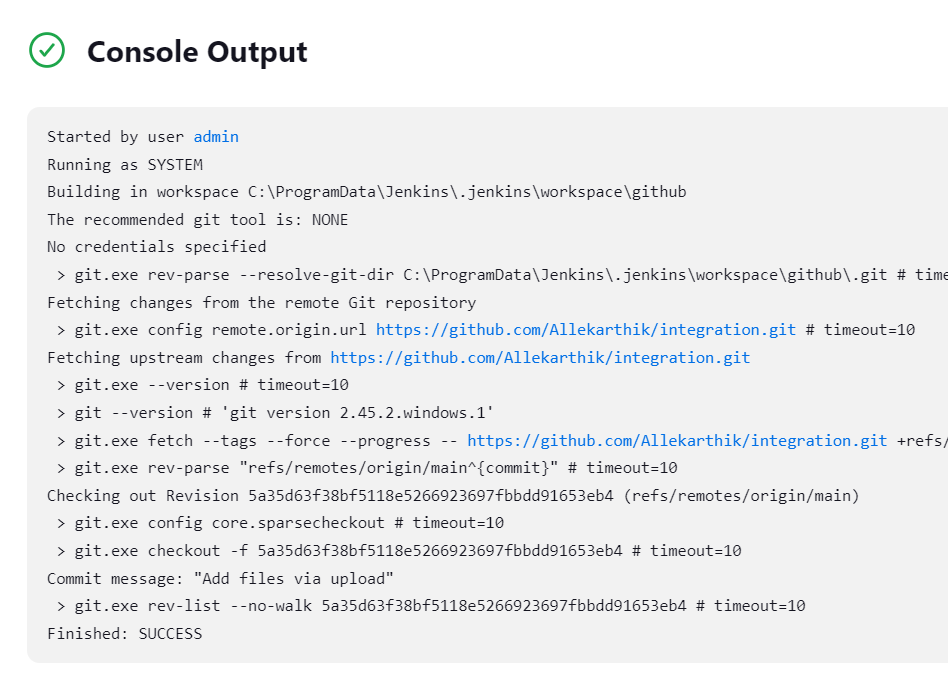


Then select repository browser as githubweb



Then tick the github hook trigger for GITScm polling

Then output will be displayed on the screen .



**Steps to build pipeline:**

Create 3 jobs like job1,job2,job3

We can create by new item->name->apply->save

Go to manage jeenkins->pulgins->available pulgins->then install build pipeline

Afer successful installation of pipeline

U can find “+” in main page click on it

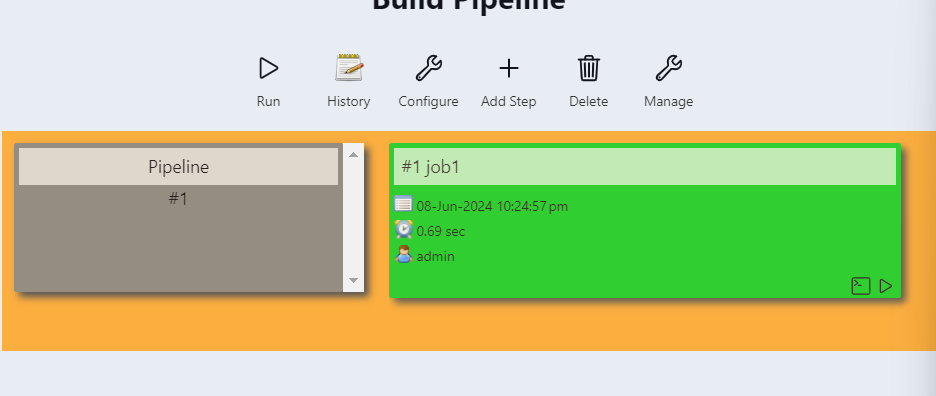
Give any name eg:karthik - > select build pipeline view - > create

Then select the initial job eg: job1



Then click on apply -> ok

Then click on run



**6. Explore Docker commands for content management**.

Docker is a powerful platform for developing, shipping, and running applications in containers. Content management within Docker involves managing images, containers, volumes, and networks. Here are some essential Docker commands for content management

**Docker Commands for Content Management**

**1. Docker run**

* **Description:** Runs a command in a new container. It’s one of the most used Docker commands because it creates and starts a new container.
* **Syntax:** docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
* **Example:** $ docker run --name mycontainer -it ubuntu:16.04 /bin/bash
  + **Explanation:**
    - --name mycontainer: Assigns the name "mycontainer" to the container.
    - -it: Combines -i (interactive) and -t (pseudo-TTY) options to keep the container running interactively.

**2. Docker start**

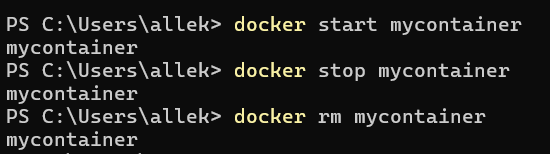
* **Description:** Starts one or more stopped containers. It does not create a new container but starts an existing one.
* **Syntax:** docker start [OPTIONS] CONTAINER [CONTAINER...]
* **Example:** $ docker start mycontainer
  + **Explanation:** Starts the container named "mycontainer".

**3. Docker stop**

* **Description:** Stops one or more running containers. It sends a SIGTERM signal to the main process inside the container, allowing it to exit gracefully.
* **Syntax:** docker stop [OPTIONS] CONTAINER [CONTAINER...]
* **Example:** $ docker stop mycontainer
  + **Explanation:** Stops the container named "mycontainer".

**4. Docker rm**

* **Description:** Removes one or more containers. The container must be stopped before it can be removed.
* **Syntax:** docker rm [OPTIONS] CONTAINER [CONTAINER...]
* **Example:** $ docker rm mycontainer
  + **Explanation:** Removes the container named "mycontainer".

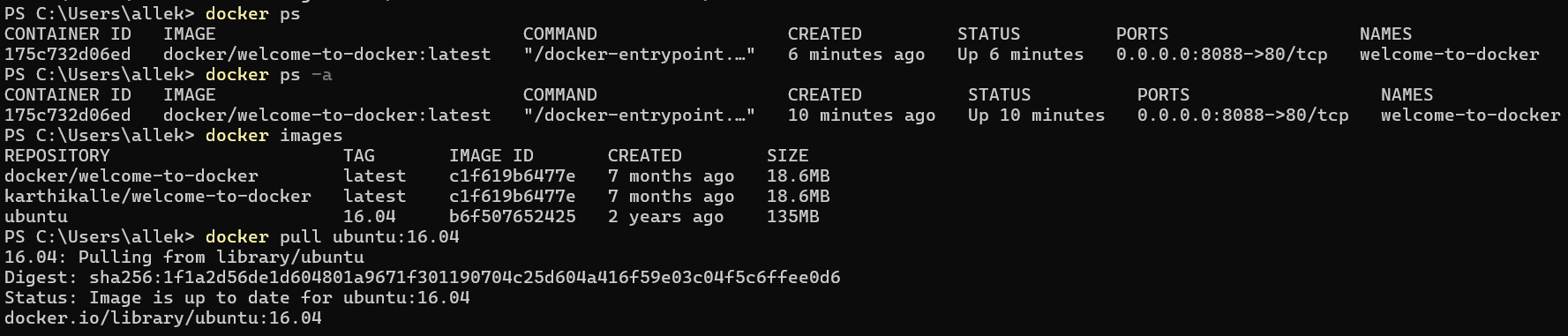


**5. Docker ps**

* **Description:** Lists containers. By default, it shows only running containers.
* **Syntax:** docker ps [OPTIONS]
* **Example:** $ docker ps
  + **Explanation:** Lists all currently running containers. To list all containers, including stopped ones, use docker ps -a.

**6. Docker images**

* **Description:** Lists images. It shows all the Docker images available on the local host.
* **Syntax:** docker images [OPTIONS] [REPOSITORY[:TAG]]
* **Example:** $ docker images
  + **Explanation:** Lists all images stored locally on the host.

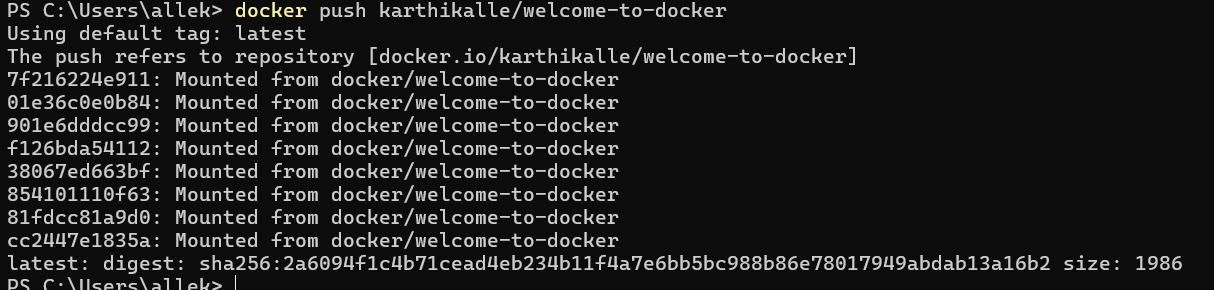


**7. Docker pull**

* **Description:** Pulls an image or a repository from a registry. It downloads the image from a Docker registry like Docker Hub.
* **Syntax:** docker pull [OPTIONS] NAME[:TAG|@DIGEST]
* **Example:** $ docker pull ubuntu:16.04
  + **Explanation:** Pulls the Ubuntu 16.04 image from the Docker Hub registry.

**8. Docker push**

* **Description:** Pushes an image or a repository to a registry. It uploads the image to a Docker registry.
* **Syntax:** docker push [OPTIONS] NAME[:TAG]
* **Example:** $ docker push myimage
  + **Explanation:** Pushes the image named "myimage" to the Docker Hub registry.



**7. Develop a simple containerized application using Docker.**

Here's an example of how you can develop a simple containerized application using Docker: Choose an application:

Before that create a folder in your file manager eg: 22507\_Docker ->python\_image ->Dockerfile ,app.py

Note:Docker should be opened first

• Choose a simple application that you want to containerize. For example, a Python script that prints "Hello World".

• Create a file named "Dockerfile" in the same directory as the application. In the Dockerfile, specify the base image, copy the application into the container, and specify the command to run the application.

Here's an example Dockerfile for a Python script:

Dockerfile

# myfirstprogram

FROM python

WORKDIR /app

COPY . /app

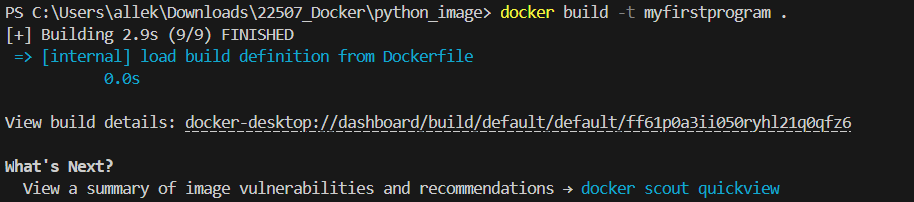
CMD ["python3", "app.py"]

app.py

print("hello world")

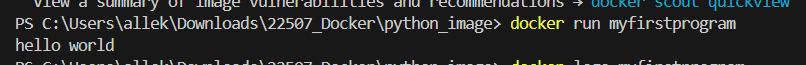
• Build the Docker image: Run the following command to build the Docker image: $ docker build -t myfirstprogram .



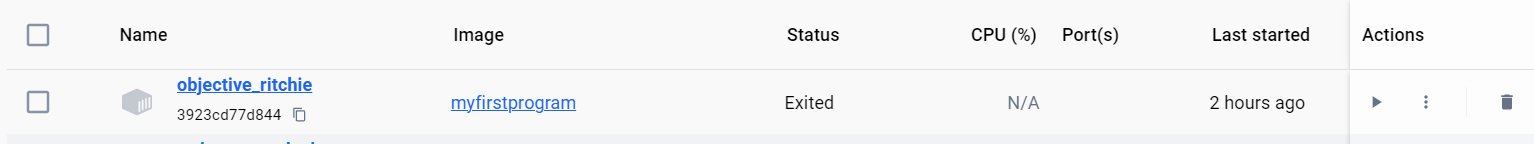


This command builds a new Docker image using the Dockerfile and tags the image with the name "myfirstprogram".

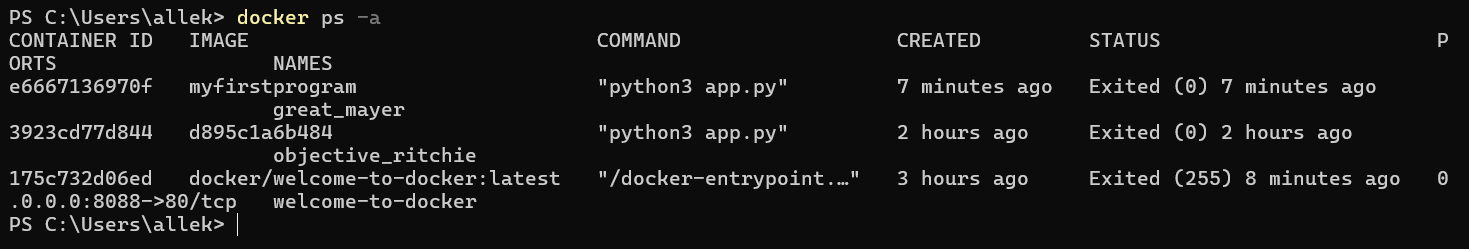
•Run the Docker container: Run the following command to start a new container based on the image: $ docker run – myfirstprogram



You can see our created container in docker apploication



You can check in Windows powershell also



This is a simple example of how you can use Docker to containerize an application. In a real-world scenario, you would likely have more complex requirements, such as running multiple containers, managing network connections, and persisting data. However, this example should give you a good starting point for using Docker to containerize your applications.

**8. Install and Explore Selenium for automated testing or Write a simple program in JavaScript and perform testing using Selenium**.

Prerequired:

Download and install Node.js

Download and install vs code

Install selenium-webdriver and install mocha

<https://storage.googleapis.com/chrome-for-testing-public/125.0.6422.141/win64/chrome-win64.zip>

->for installing selenium web driver go to any web browser->selenium web driver install ->Download selenium->javascript stable 4.21->chromedriver.exe->win64

Steps:

Create a folder called newfolder in your downloads /or any other main folder

Open this folder in vs code and go to termina->new terminal then type the below command i.e npm init where it atomatically creates package.json file

{

  "name": "new-folder",

  "version": "1.0.0",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "author": "",

  "license": "ISC",

  "description": "",

  "dependencies": {

    "selenium-webdriver": "^4.21.0"

  },

  "devDependencies": {

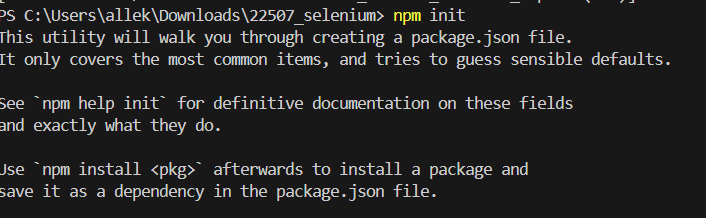
    "chromedriver": "^125.0.3",

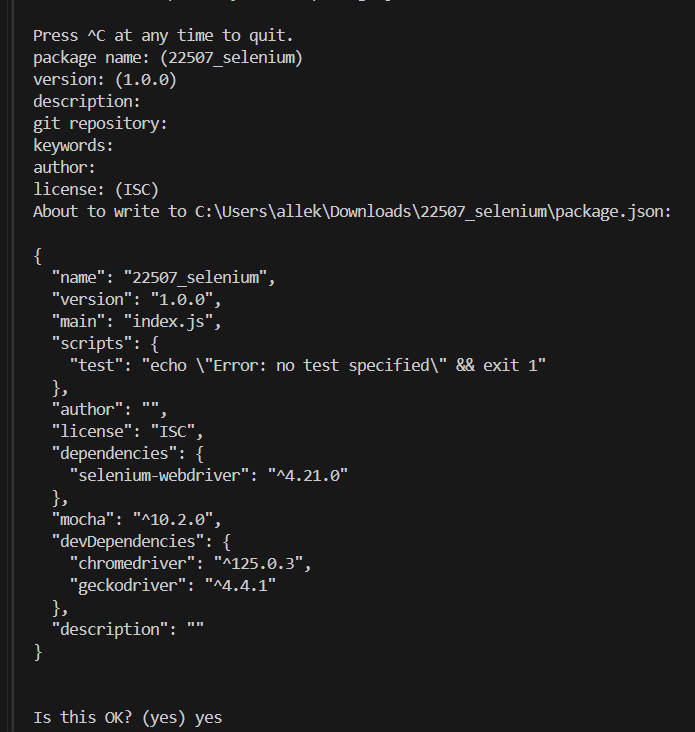
    "geckodriver": "^4.4.1"

  },

  "mocha":"^10.2.0" //here u need to add this extra line

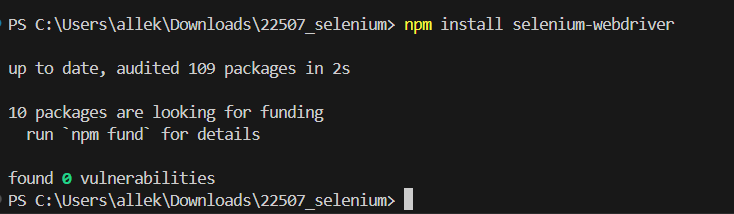
}



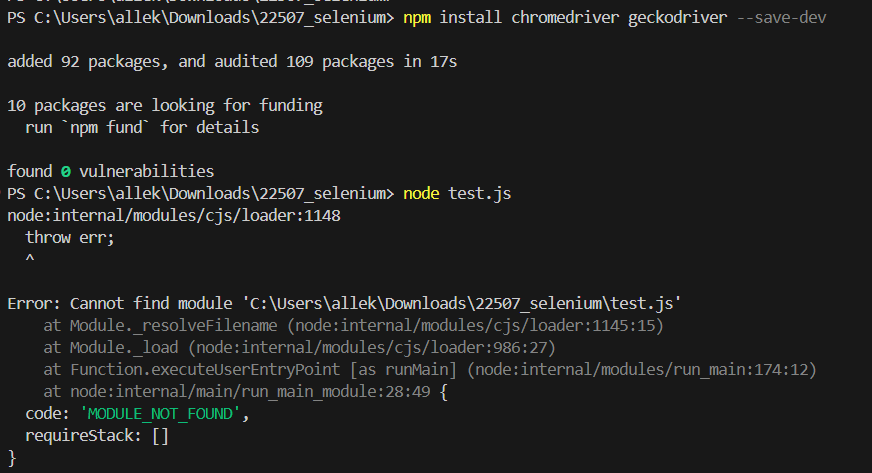


Then next step is to install selenium

After installing it will create an package-lock.json file



After succesfull installation of selenium we need to install chromedriver



Home.js

const { Builder, By, Key, until } = require('selenium-webdriver');

const chrome = require('selenium-webdriver/chrome');

(async () => {

const driver = await new Builder()

.forBrowser('chrome')

.setChromeOptions(new chrome.Options())

.build();

try {

await driver.get('https://www.google.com');

await driver.findElement(By.name('q')).sendKeys('Selenium', Key.RETURN);

await driver.wait(until.titleContains('Selenium'), 100000000000);

} catch (error) {

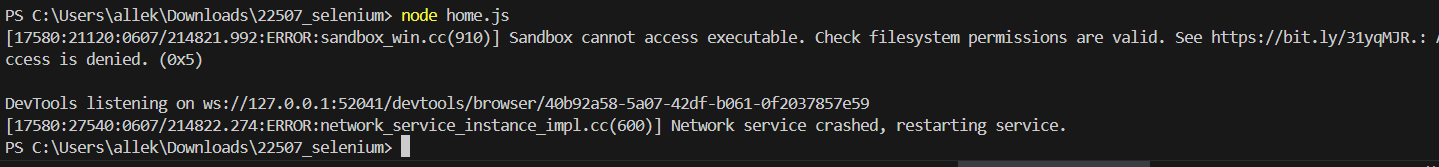
console.error('Test failed:', error);

} finally {

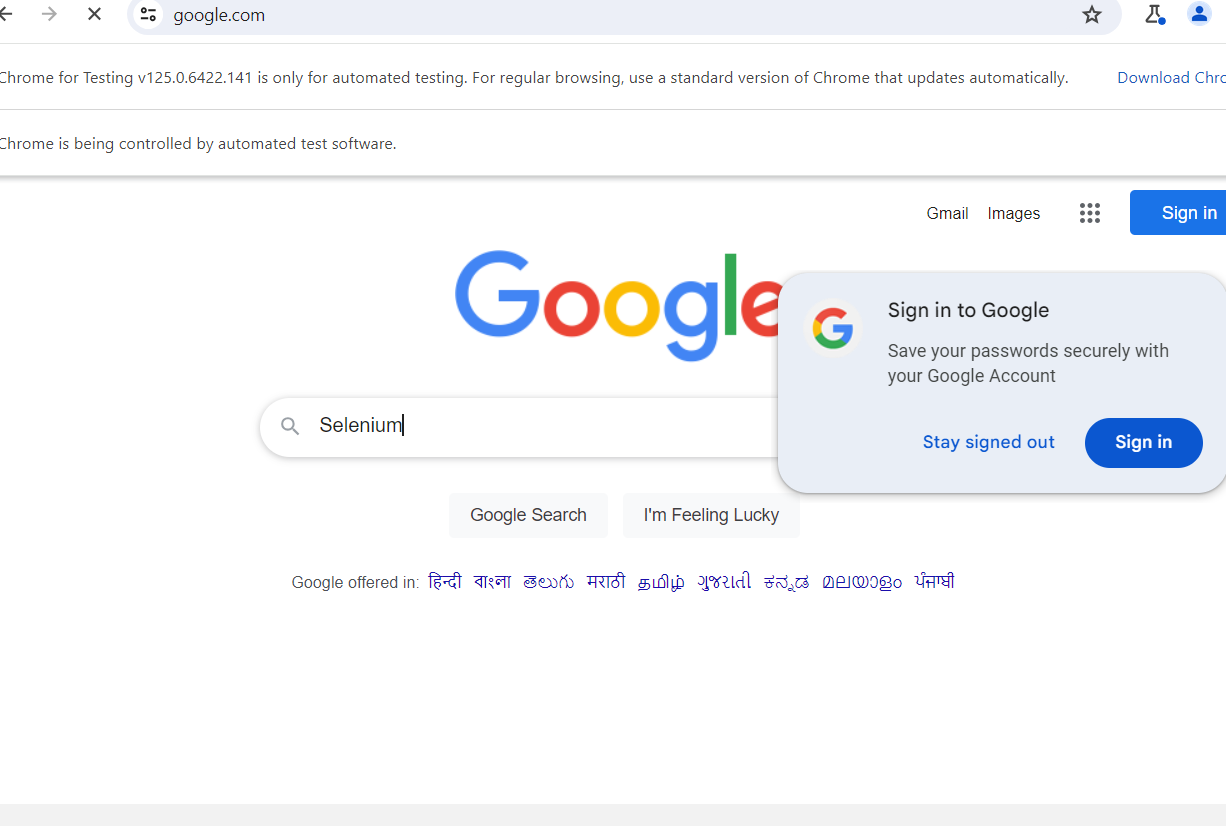
await driver.quit();

}

})();



Output:



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

const { Builder, By, Key, until } = require('selenium-webdriver');

const chrome = require('selenium-webdriver/chrome');

(async () => {

const driver = await new Builder()

.forBrowser('chrome')

.setChromeOptions(new chrome.Options())

.build();

try {

// Test Case 1: Navigate to Google and verify title

await driver.get('https://www.google.com');

await driver.wait(until.titleContains('Google'), 10000);

console.log('Test Case 1 Passed: Title contains "Google"');

// Test Case 2: Search for "Selenium" on Google

await driver.findElement(By.name('q')).sendKeys('Selenium', Key.RETURN);

await driver.wait(until.titleContains('Selenium'), 10000);

console.log('Test Case 2 Passed: Title contains "Selenium"');

// Test Case 3: Verify search results

const searchResults = await driver.findElements(By.css('div.g'));

console.log(`Test Case 3 Passed: Found ${searchResults.length} search results`);

// Test Case 4: Verify the presence of the search input box

const searchInput = await driver.findElement(By.name('q'));

const isSearchInputDisplayed = await searchInput.isDisplayed();

console.log(`Test Case 4 Passed: Search input box is displayed: ${isSearchInputDisplayed}`);

} catch (error) {

console.error('One or more test cases failed:', error);

} finally {

await driver.quit();

}

})();

