# devops project

Git Collaboration Theory – Overview

Git is a distributed version control system (DVCS) that allows multiple developers to collaborate efficiently on a project, whether working on the same files or different parts of the codebase. Here's a breakdown of the core theory behind Git collaboration:

Ç 1. Distributed System

* Every developer has a full copy of the repository including the history.
* Work can continue ofline, and changes are merged later.

’⬛ 2. Branches

* A branch is a pointer to a snapshot of changes.
* The main (or master) branch is usually the stable production version.
* Developers create feature branches to work on new functionality independently.
* Enables parallel development without conflicts.

3. Commits and History

* A commit is a snapshot of changes along with a message.
* Commits form a linear or branching history, enabling tracking and rollback.

.çv† 4. Common Commands for Collaboration

|  |  |  |
| --- | --- | --- |
| Action | Git Command | Description |
| Clone repo | git clone <url> | Copy a remote repo locally |
| Create branch | git checkout -b feature-x | Start new feature work |
| Stage changes | git add . or git add <file> | Prepare files for commit |
| Commit changes | git commit -m "message" | Save snapshot |
| Push changes | git push origin branch-name | Send changes to remote |
| Pull updates | git pull | Fetch + merge remote changes |
| Merge branch | git merge feature-x | Combine branches |

|  |  |  |
| --- | --- | --- |
| Action | Git Command | Description |
| Resolve conflicts | Manual + git add + git commit | When Git can’t auto-merge |

~¸’× ‘¸˛ 5. Collaboration Workflow

* Centralized Workflow (Small teams)
  + Everyone works on the main branch.
  + Pull before push to avoid overwriting.
* Feature Branch Workflow (Common)
  + Create a new branch per feature or bug.
  + Submit Pull Requests (PR) or Merge Requests (MR).
  + Peer reviews, test, then merge into main.
* Fork & Pull Workflow (Open Source)
  + Fork the original repo → clone → make changes.
  + Push to your fork, then create a PR to the main project.

˛C\* 6. Conflict Resolution

* + When multiple people edit the same lines, Git can't auto-merge.
  + Developer must resolve conflicts manually, then commit the result.

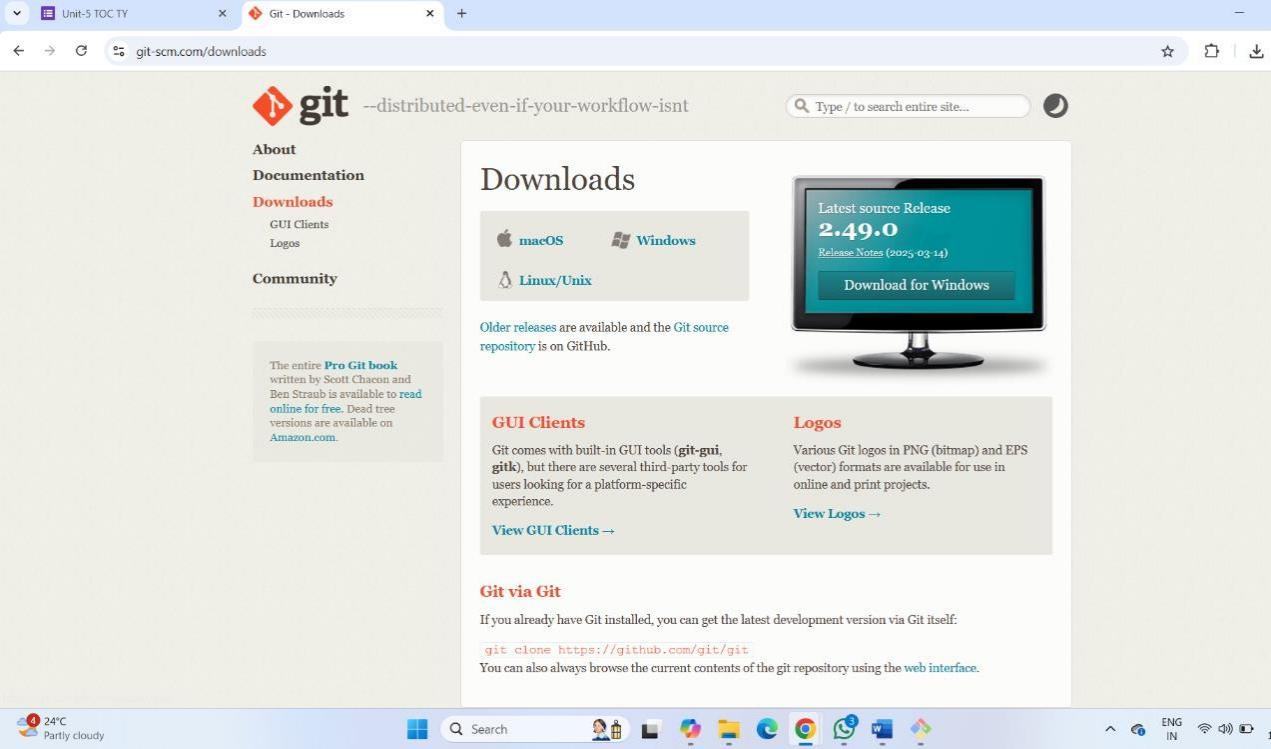
t.º 7. Access Control

* Permissions on repositories (read/write/admin).
* Roles: Owner, Maintainer, Contributor, etc.

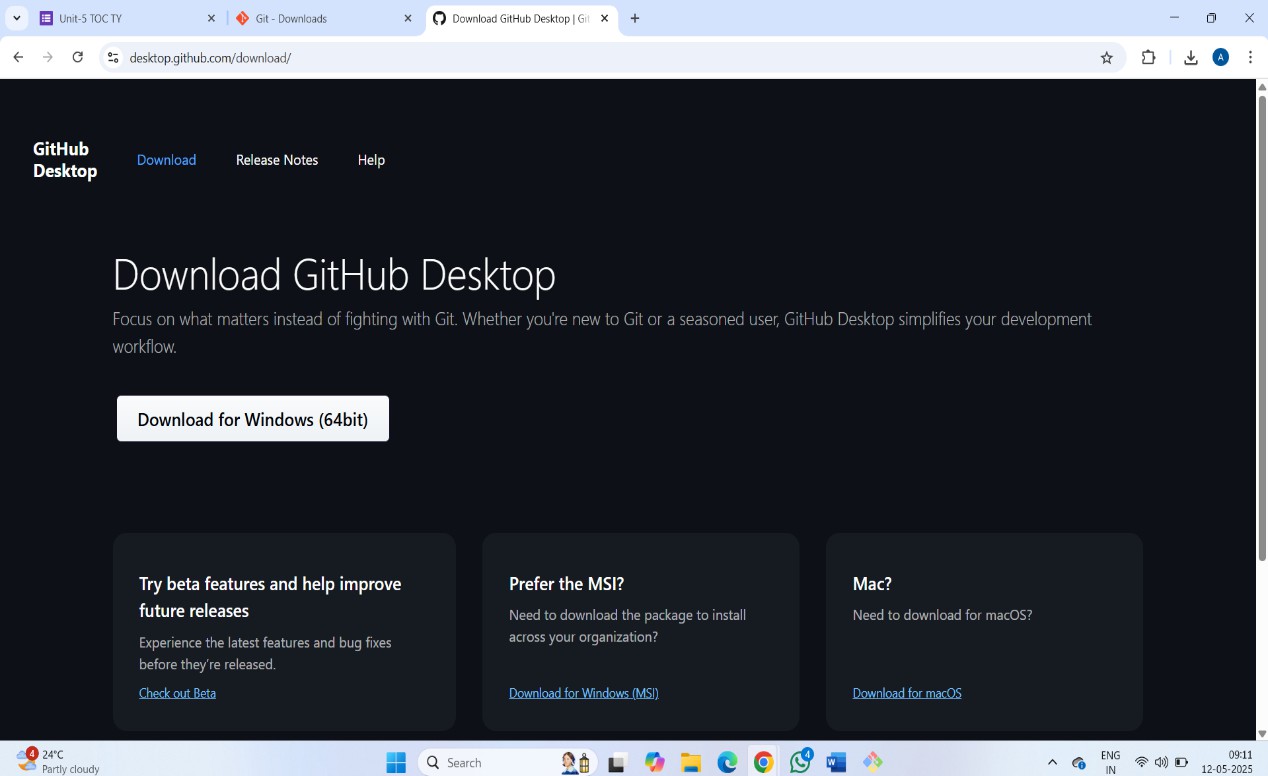
` ^ 8. CI/CD Integration

* Git is often integrated with tools like GitHub Actions, GitLab CI, Jenkins, etc.
* Ensures code is tested and deployed automatically after merging.

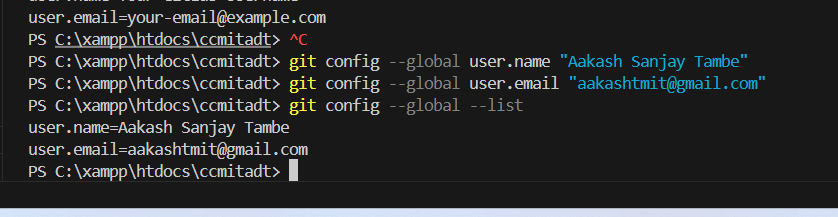
# TASK- 1

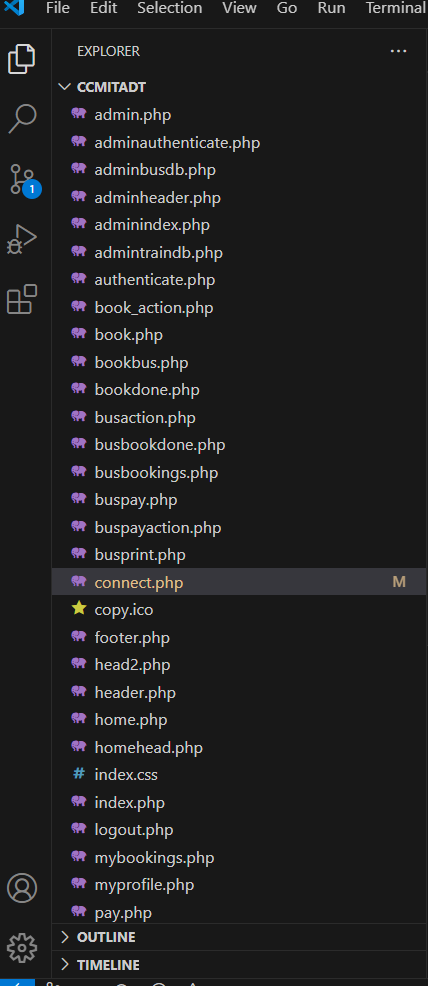
**\*)Download git and git desktop**

**\*)**

****

# Open gitbash and set globally variable



**\*) upload project on GitHub using git commands**

\*)PS C:\xampp\htdocs\ccmitadt> git init

Reinitialized existing Git repository in C:/xampp/htdocs/ccmitadt/.git/ PS C:\xampp\htdocs\ccmitadt> git remote -v

origin https://github.com/assh/projectdevops.git (fetch) origin https://github.com/assh/projectdevops.git (push)

PS C:\xampp\htdocs\ccmitadt> git remote set-url origin https://github.com/assh/projectdevops.git

PS C:\xampp\htdocs\ccmitadt> git remote remove origin PS C:\xampp\htdocs\ccmitadt> git remote add origin <https://github.com/assh>

/projectdevops.git

PS C:\xampp\htdocs\ccmitadt> git push -u origin main info: please complete authentication in your browser... Enumerating objects: 58, done.

Counting objects: 100% (58/58), done. Delta compression using up to 8 threads

Compressing objects: 100% (47/47), done.

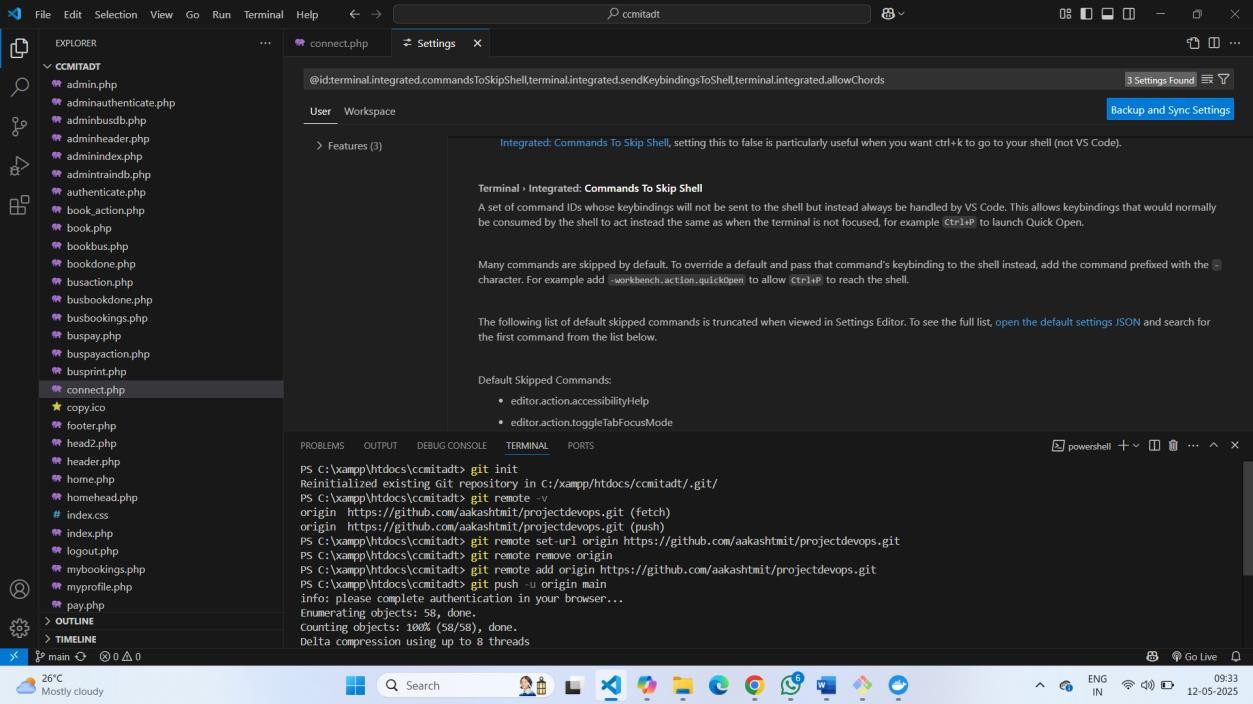
Writing objects: 100% (58/58), 49.53 KiB | 9.91 MiB/s, done. Total 58 (delta 10), reused 54 (delta 9), pack-reused 0 (from 0)

remote: Resolving deltas: 100% (10/10), done.

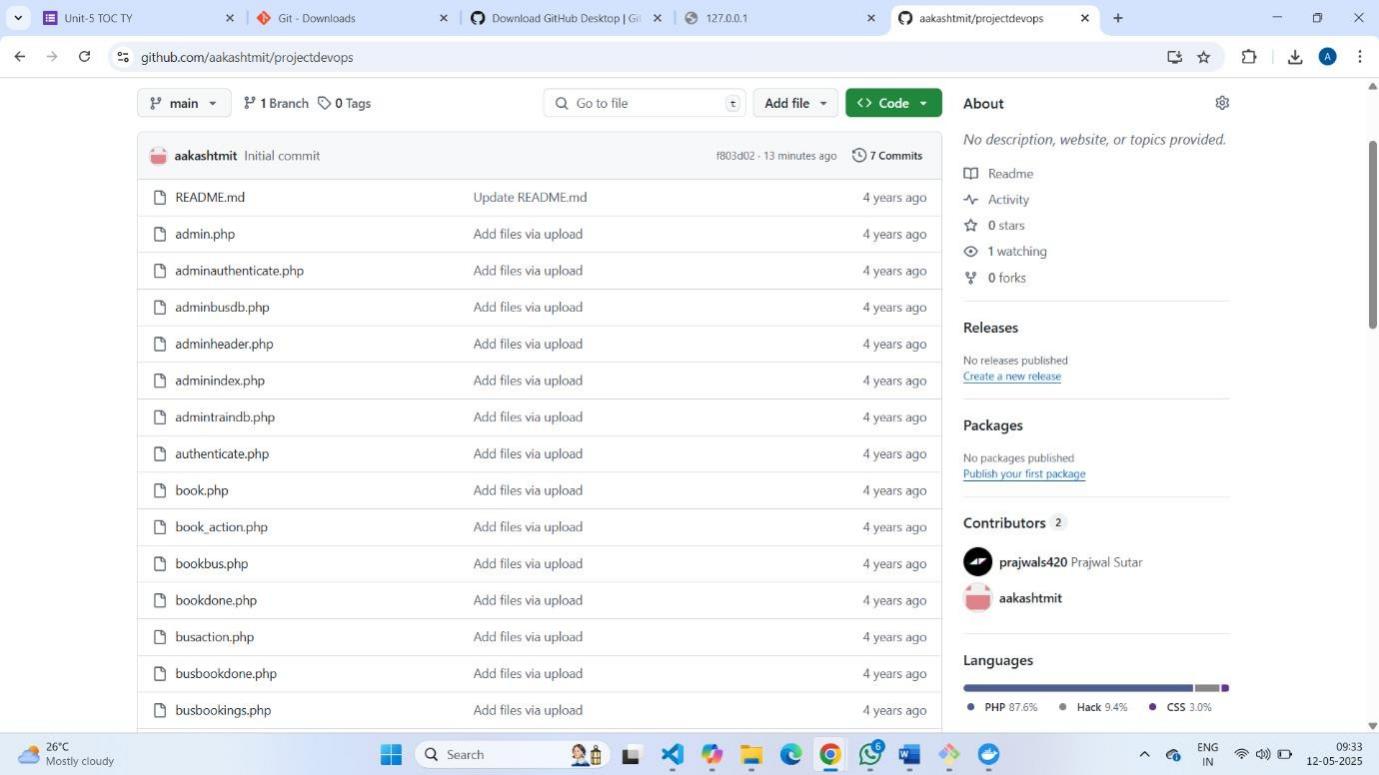
To https://github.com/aakashtmit/projectdevops.git

\* [new branch] main -> main

branch 'main' set up to track 'origin/main'. PS C:\xampp\htdocs\ccmitadt>



\*) push in GitHub



\*) project success fully uploaded

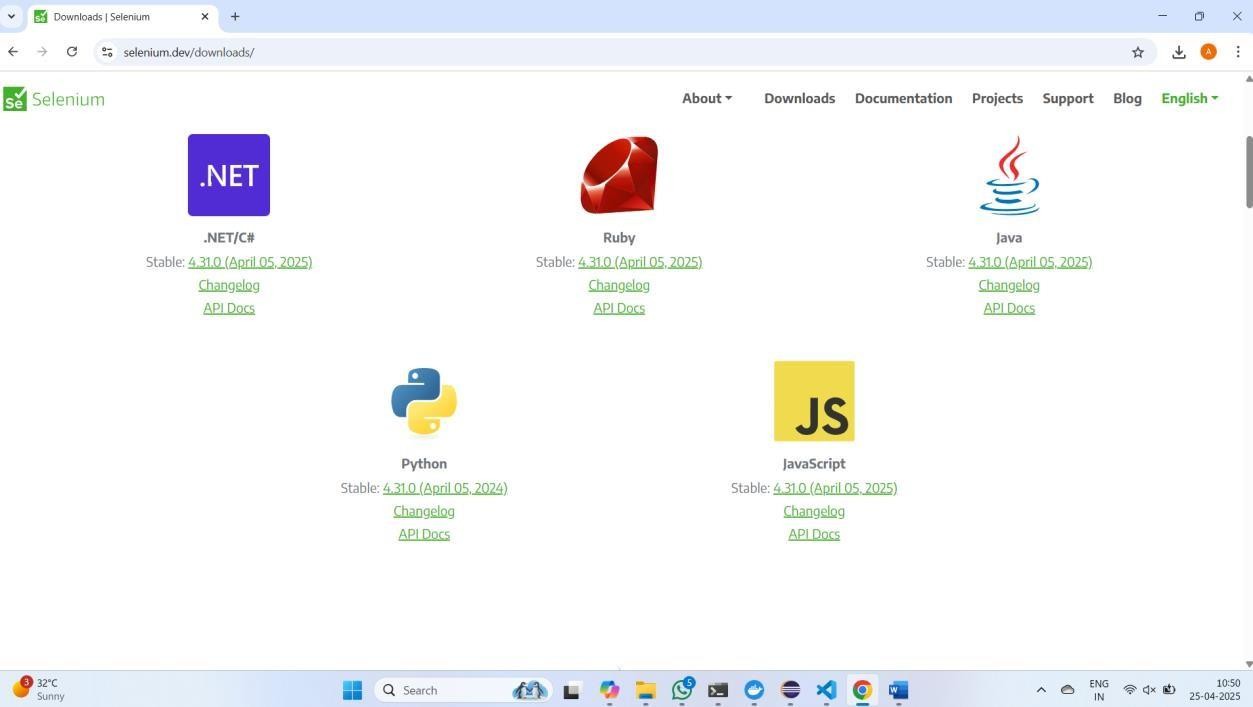
**TASK 2 - TEST IN JAVA WITH NG**

TA2

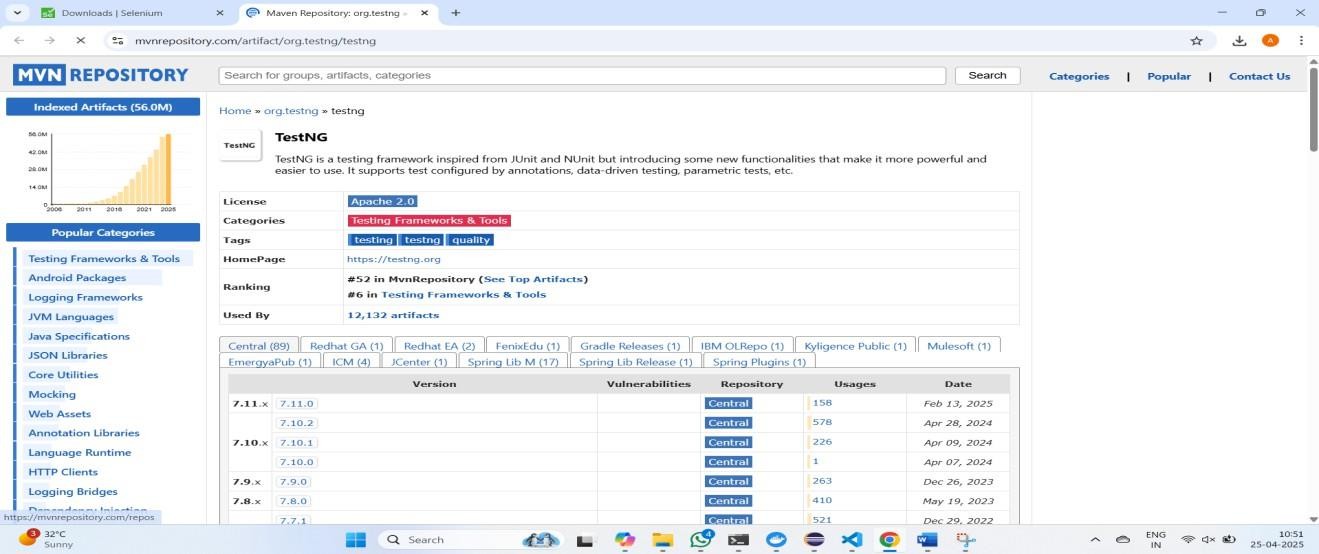
QUESTION 3 –

STEP 1 –

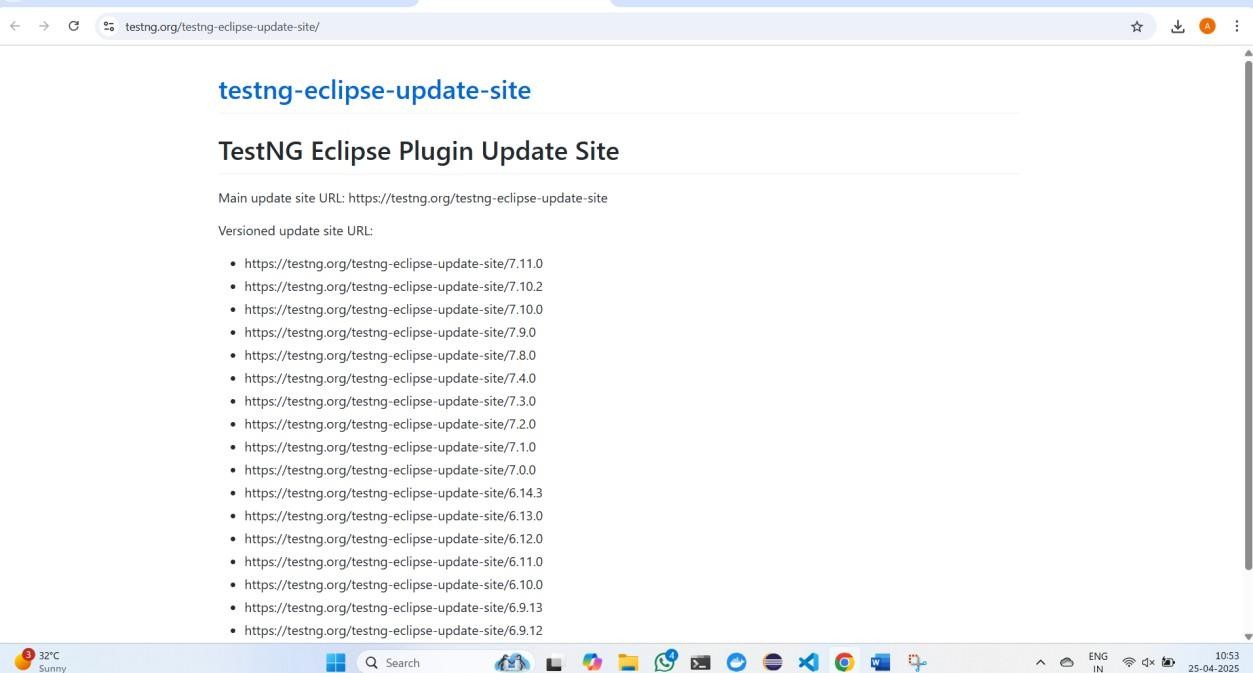
INSTALLATION OF SELENUIM JAR FILE FROM OFFICAL WEBSITE



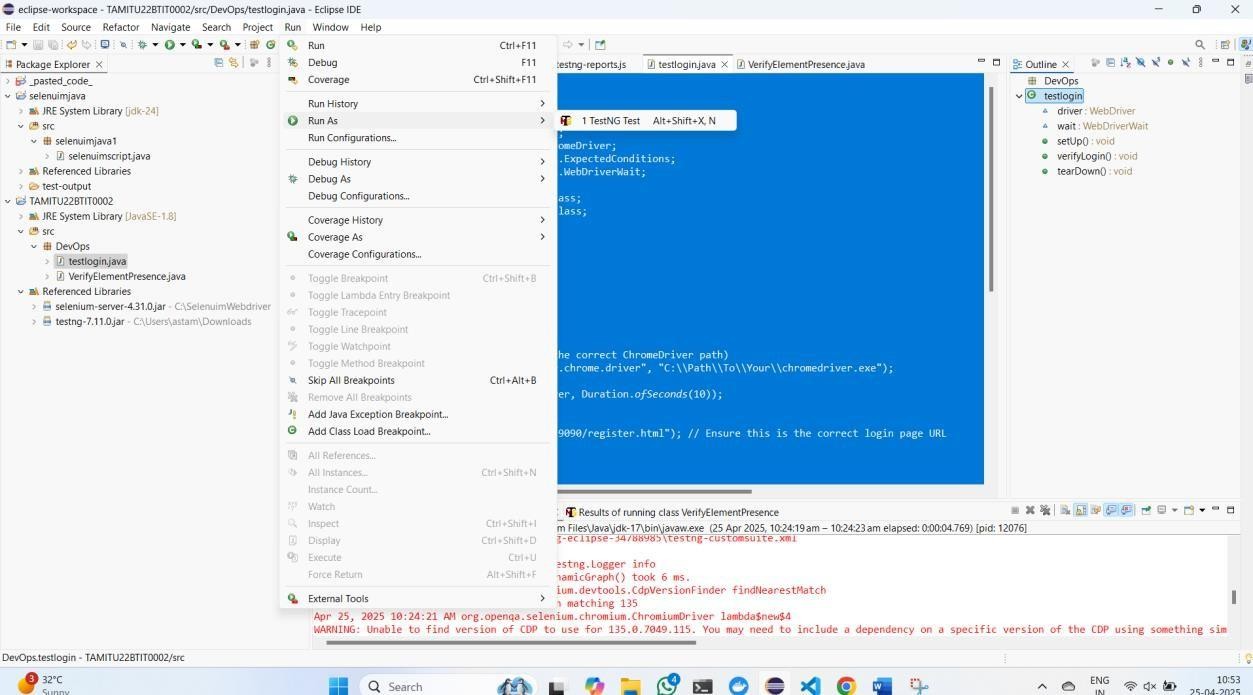
STEP 2- DOWNLOAD MAVEN REPO



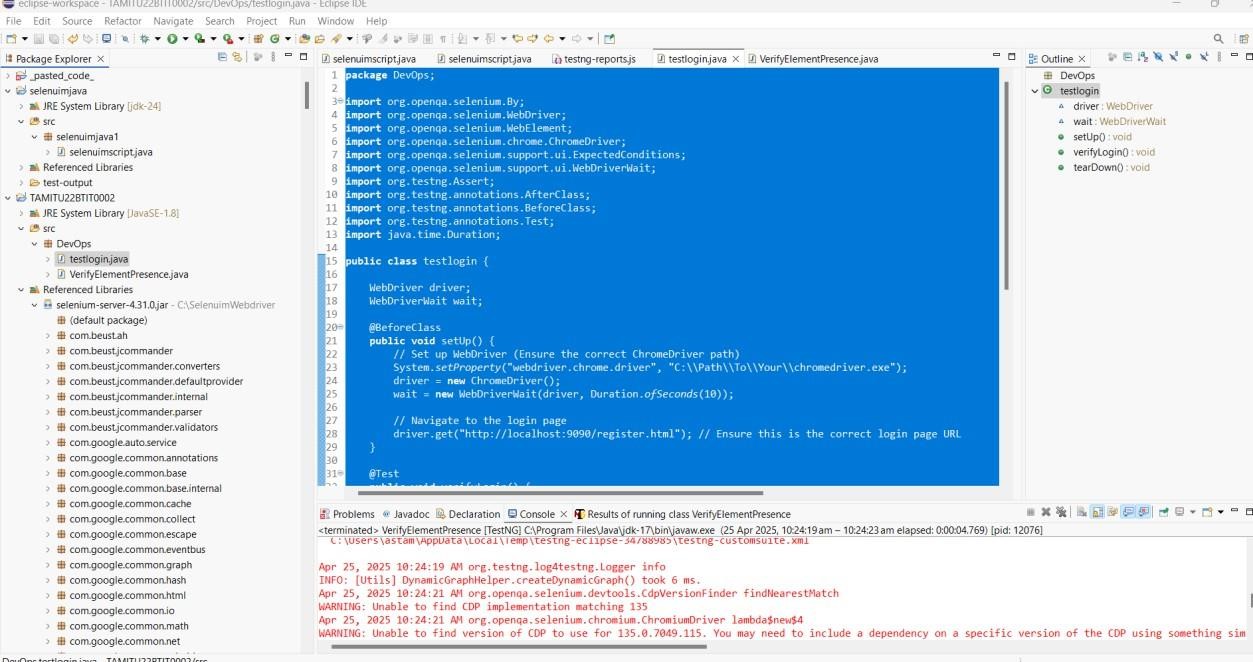
STEP 3 -OPEN ECLIPSE AND INSTALL TEST NG FROM SOFTWARE



Step 4 –

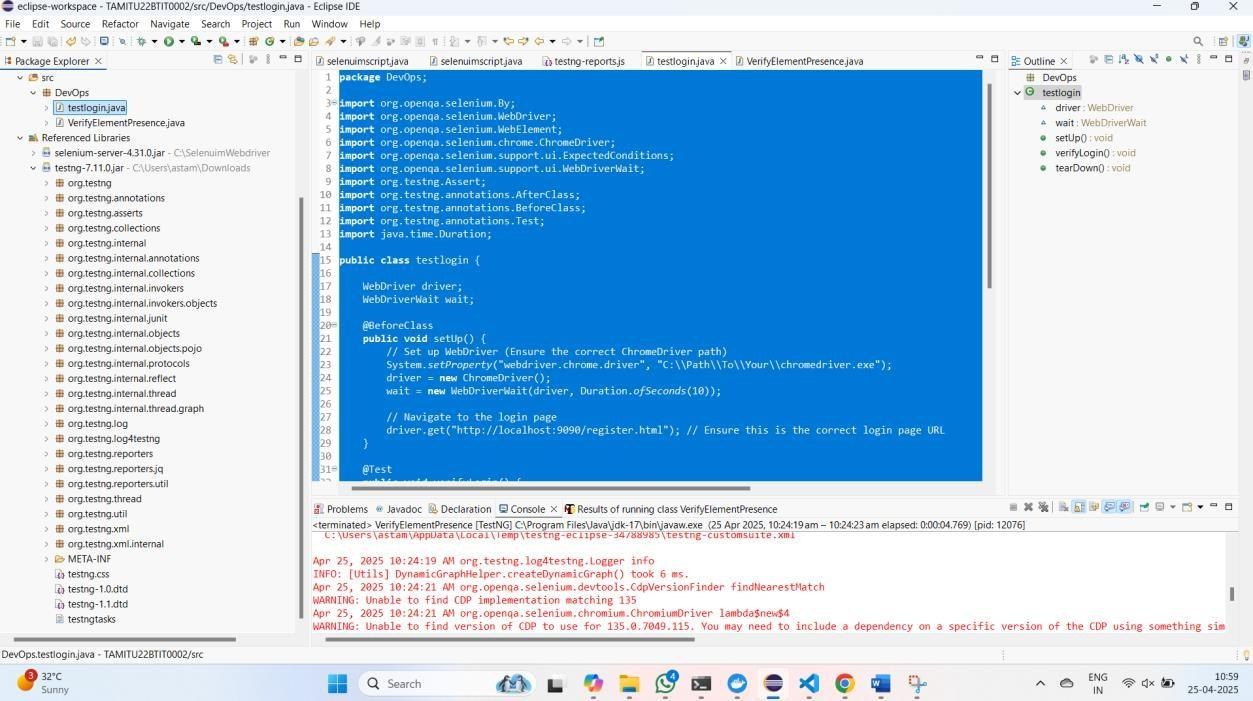


We successfully install test ng



Installation of libraby jar file

Step 6



Install lation of maven testing libray

Code for logon test case

**package** DevOps;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.support.ui.ExpectedConditions; **import** org.openqa.selenium.support.ui.WebDriverWait; **import** org.testng.Assert;

**import** org.testng.annotations.AfterClass; **import** org.testng.annotations.BeforeClass; **import** org.testng.annotations.Test; **import** java.time.Duration;

**public class** testlogin {

WebDriver driver; WebDriverWait wait;

@BeforeClass

**public void** setUp() {

// Set up WebDriver (Ensure the correct ChromeDriver path) System.*setProperty*("webdriver.chrome.driver",

"C:\\Path\\To\\Your\\chromedriver.exe"); driver = **new** ChromeDriver();

wait = **new** WebDriverWait(driver, Duration.*ofSeconds*(10));

// Navigate to the login page driver.get("<http://localhost:9090/register.html>"); // Ensure this is the

correct login page URL

}

@Test

**public void** verifyLogin() {

// Locate login elements WebElement emailField =

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*id*("emailAddress"))); WebElement passwordField = wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*id*("loginPassword")));

// Try an alternative method to locate the Login button WebElement loginButton =

wait.until(ExpectedConditions.*elementToBeClickable*(By.*xpath*("//button[contains(tex t(),'Login')]")));

// Enter login credentials emailField.sendKeys("[testuser@example.com](mailto:testuser@example.com)"); passwordField.sendKeys("SecurePass123"); loginButton.click();

// Verify login success with an updated selector WebElement confirmationMessage =

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*cssSelector*(".confirma tion-class"))); // Update based on actual site

// Assertion to validate login success Assert.*assertTrue*(confirmationMessage.isDisplayed(), "Login verification

failed!");

System.***out***.println("Login successful!");

}

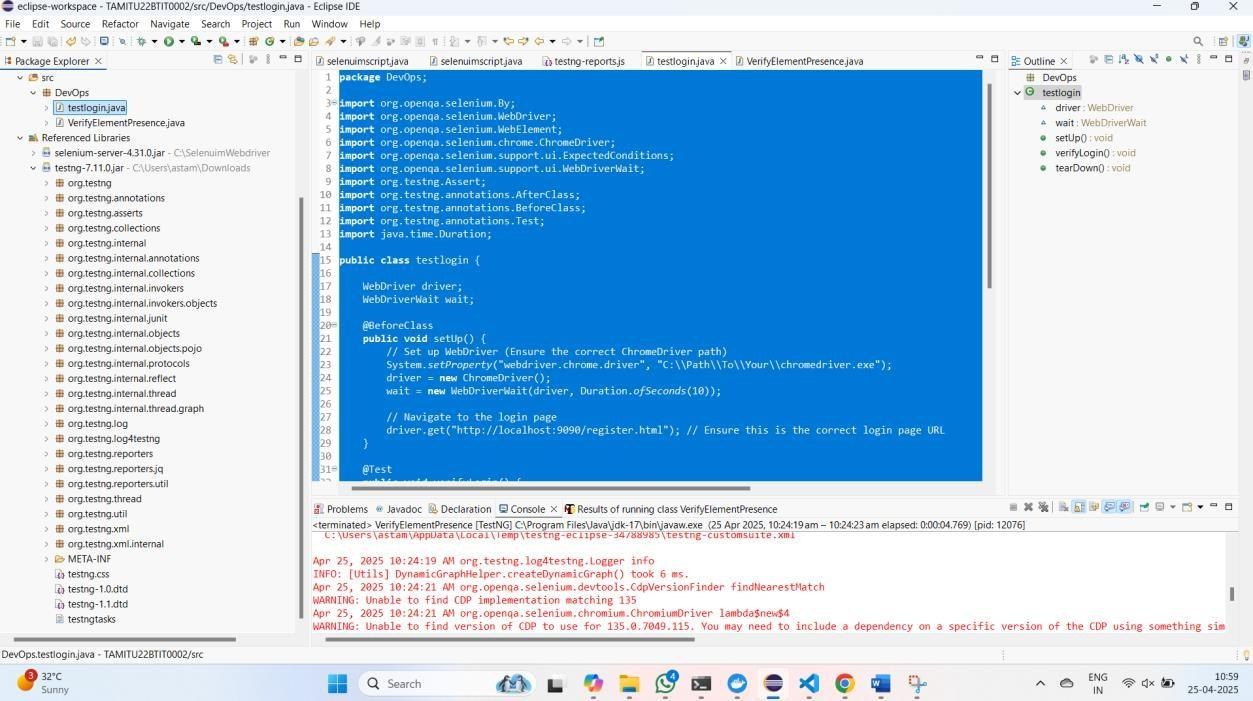
@AfterClass

**public void** tearDown() {

// Close the browser driver.quit();

}

}



Test case 2 elemetn present or not

**package** DevOps;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.testng.Assert;

**import** org.testng.annotations.AfterClass; **import** org.testng.annotations.BeforeClass; **import** org.testng.annotations.Test;

**import** java.time.Duration;

**public class** VerifyElementPresence { WebDriver driver;

@BeforeClass

**public void** setUp() {

// Set up ChromeDriver (Ensure correct path)

// System.setProperty("webdriver.chrome.driver", "C:\\Path\\To\\Your\\chromedriver.exe");

driver = **new** ChromeDriver(); driver.manage().timeouts().implicitlyWait(Duration.*ofSeconds*(10)); driver.get("<http://localhost:9090/register.html>"); // Update to the

correct URL

}

@Test

**public void** testElementPresence() {

**try** {

WebElement element = driver.findElement(By.*xpath*("//\*[@id='main- wrapper']/div/div/div[2]/div/div[1]"));

Assert.*assertTrue*(element.isDisplayed(), "Element is \*\*not\*\*

visible!");

System.***out***.println("Element is \*\*present and displayed\*\*.");

} **catch** (Exception e) {

Assert.*fail*("Element \*\*not found\*\* on the page."); System.***out***.println("Element is \*\*not present\*\*.");

}

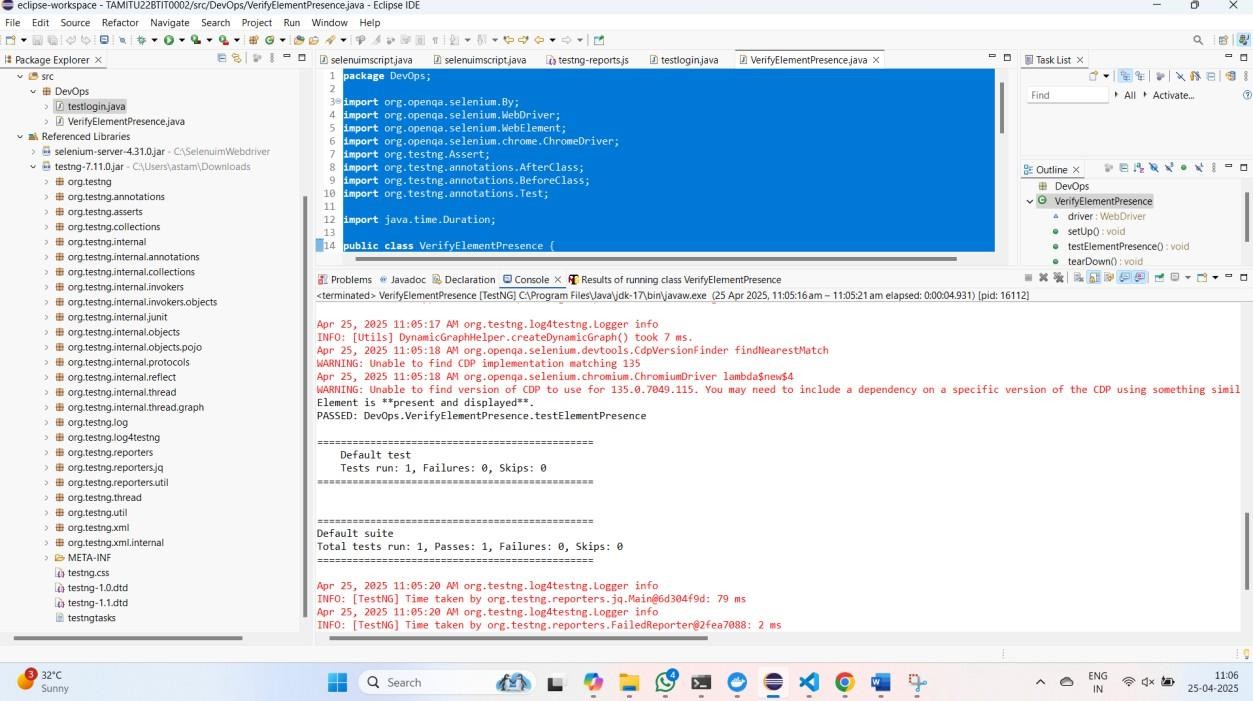
}

@AfterClass

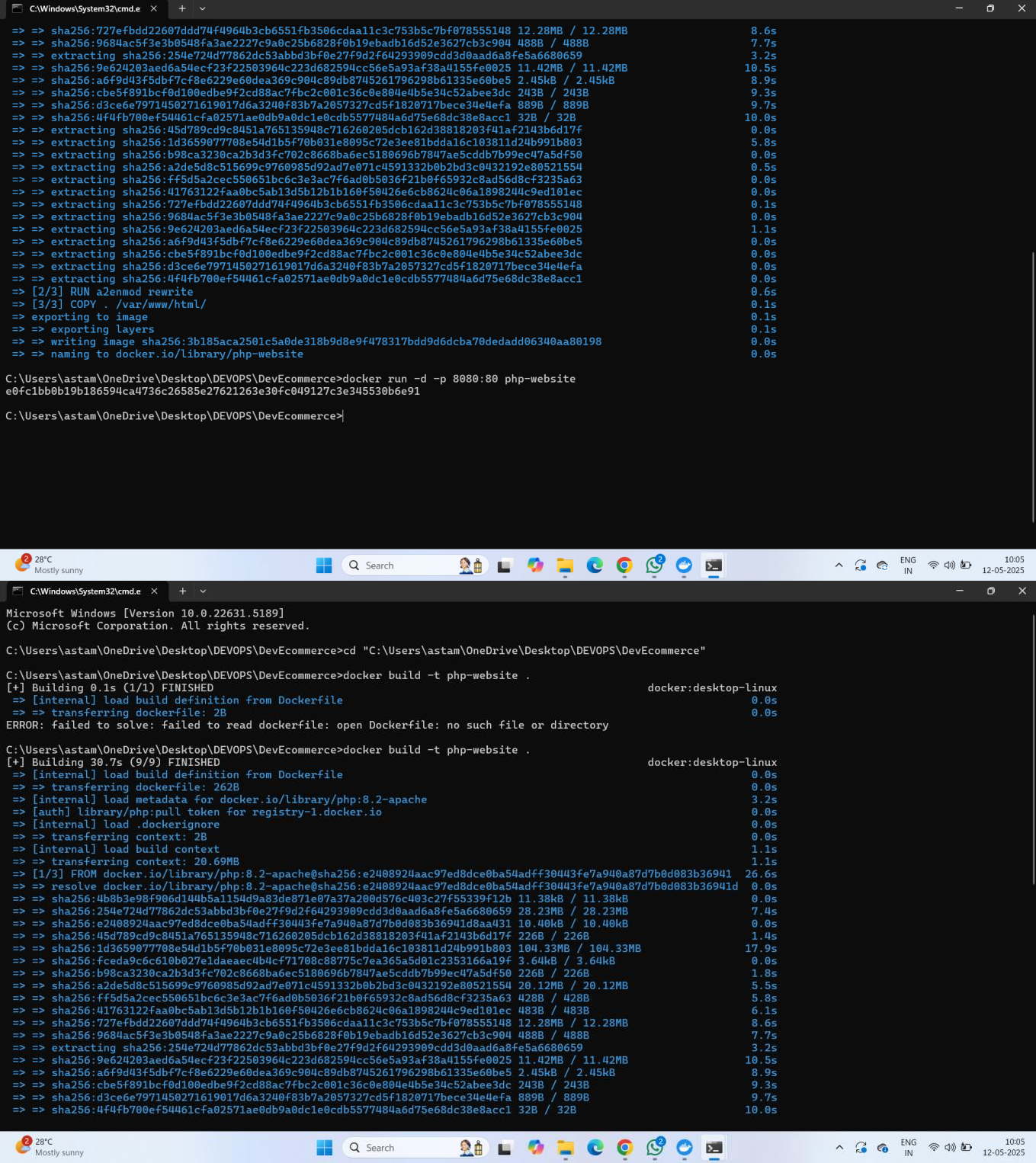
**public void** tearDown() { driver.quit();

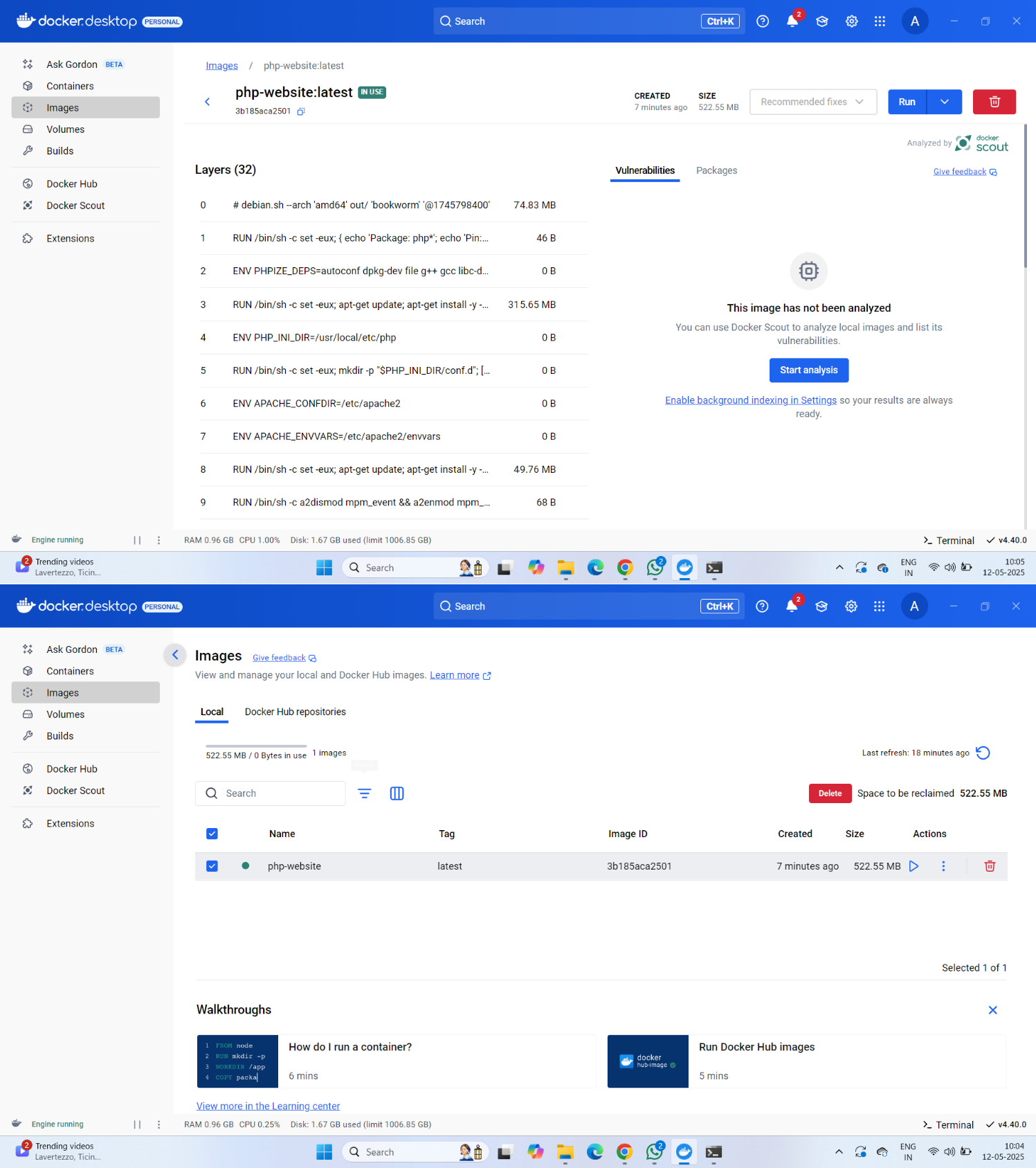
}

}



output

**TASK 3**

****

