EXPERIMENT NO: 1 **DATE:**

Registration form

Aim: Write a code for a simple user registration form for an event.

PROGRAMS:

Here, app.py should be in the folder itself whereas another 2 programs must be under the name templates file in the created folder.

App.py

```
from flask import Flask, request, render_template
app = Flask(__name__)
@app.route('/register', methods=['GET', 'POST'])
def register():      if request.method == 'POST':
name = request.form['name']          email =
request.form['email']          password =
request.form['password']          return
render_template('success.html')    return
render_template('register.html') if __name____ ==
'__main__':        app.run(debug=True)
```

Register.html

```
<!DOCTYPE html>
<html lang="en">
<head>
     <title>Registration</title>
</head>
<body>
    <h2>Registration Form</h2>
    <form method="post">
        <input type="text" name="name" placeholder="Name">
        <input type="email" name="email" placeholder="Email">
        <input type="password" name="password" placeholder="Password"> <input
type="submit" value="Submit">
    </form>
</body>
</html>
```

Success.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>RegistrationSuccess</title>
</head>
<body>
    <h2>Registration Successful</h2>
</body>
</html>
```

OUTPUT:

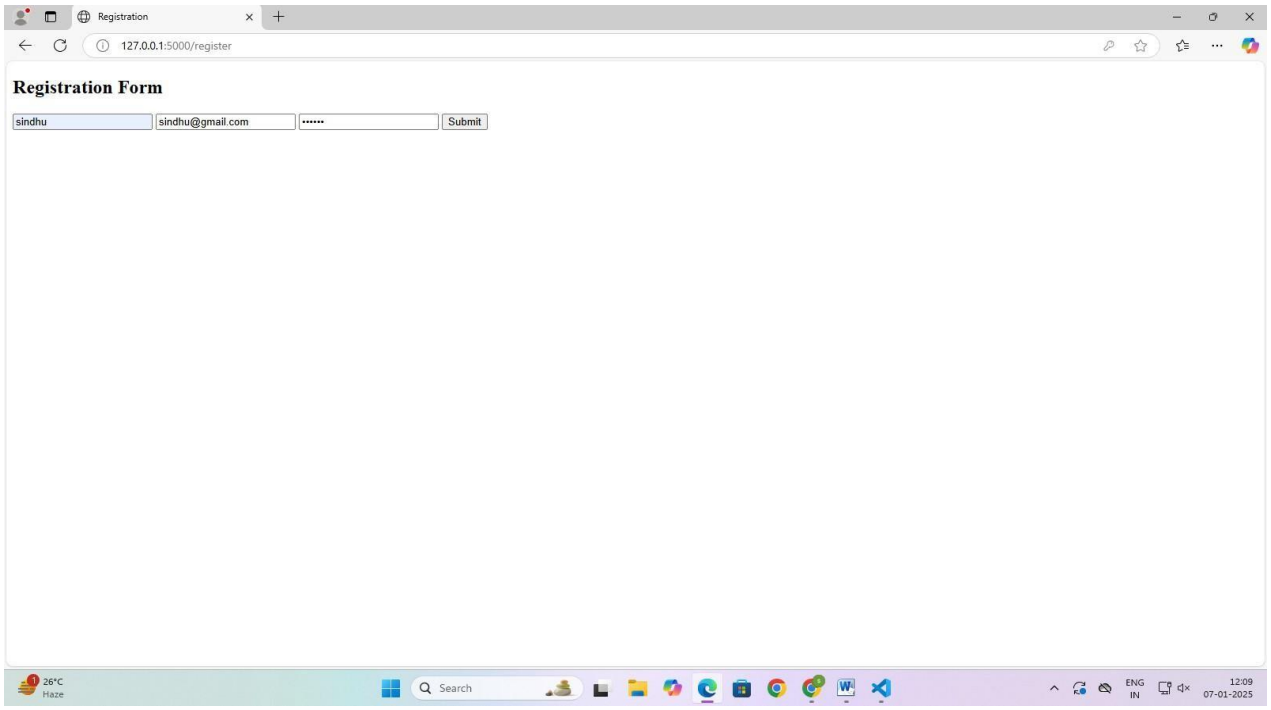
Flask run:-

First, install Flask by using the command 'pip install flask'.

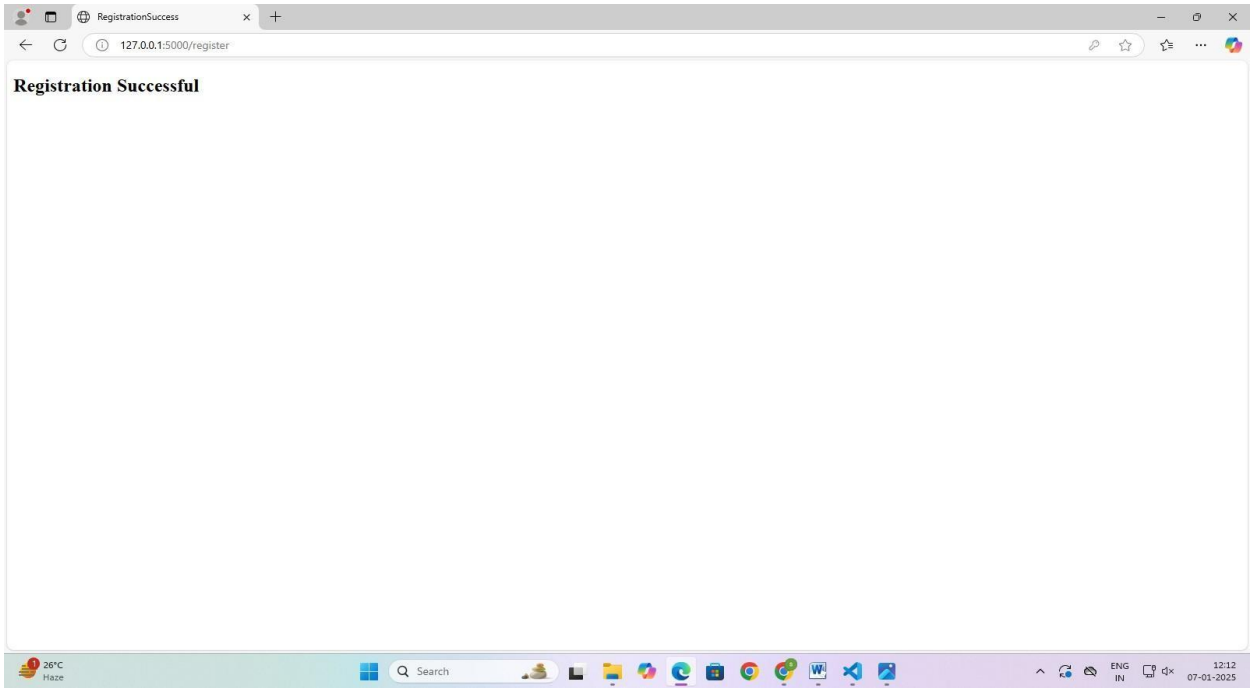
After installing Flask, run app.py by the command 'py app.py'

Next as per the image given below, there will be a port. Open that port and complete the restration process, then you'll get a success page.

Register.html :-



Success.html :-



EXPERIMENT NO: 2 **DATE:**

Git and GitHub Commands

Aim:- Explore the Git and GitHub Commands

Description:-

Git commands are crucial for efficient collaboration and project management. Now we'll explore a list of important Git commands like git commit command, git pull command, and git push command, etc., that will help improve workflow and optimize productivity. These are a list of Git Commands that can be used frequently.

The commands are:

➢ **git help:** Take help from the Git help section for different commands and other errors.

➢ **git config:** To set the basic configurations on Git like your name and email.

➢ **git init:** To create a local git repository for us in our store folder. Tis will help to manage the git commands for that particular repository.

➢ **git status:** To see what's changed since the last commit. It shows al the files that have been added and modified and are ready to be committed and untracked files.

➢ **git commit:** To commit our changes (taking a snapshot) and provide a message to remember for future reference.

➢ **git log:** To check the history of commits for our reference.

➢ **git add:** To add a specific list of files to the staging area.

➢ **git push:** To push all the contents of our local repository that belong to the master branch to the server (Global repository).

➢ **git branch:** To see all the branches present and current branches that we are working on.

➢ **git fetch:** To fetch down any changes from the global repository to the current repository.

➢ **git version:** It is used to show the current version of Git.

➢ **git checkout:** The git checkout command to switch an existing branch (or) to create and switch to a new one.

➢ **git pull:** The contents of the remote repository are fetched and integrated into your local repository using this command.

➢ **git merge:** The command git merge joins your branch to the parent branch. Depending on your local process, the present parent branch can be either a development branch or a master branch.

➢ **git clone:** To make a local copy of the global repository in your system (the git clone command downloads the repository and creates a remote named origin which can be checked by the command – git remote -v).

➢ **git remove rm:** To remove a remote from our local repository.

Conclusion:

The above Git commands are essential for managing repositories, tracking changes, and collaborating effectively. They cover setup, version control, branch management, and syncing between local and remote repositories.

EXPERIMENT NO: 3 **DATE:**

Aim:- Practice Source code management on GitHub. Experiment with the source code written in exercise 1.

Description:

GitHub is a cloud-based platform that allows developers to store, share, and collaboratively work on code using a version control system called Git, enabling them to track changes made to their projects over time, review code with others, and manage different versions of their files within a "repository" - essentially a project folder on GitHub; making it a popular tool for open-source software development and team collaboration**.**

Let us practice Source code management on GitHub by creating a repository with the name **"Sinzz0.github.io"** and by using the git commands we push the source code in the repository.

Procedure:

First, there need to be a folder named templates in which register.html, success.html should be there in it. App.py should be outside of the templates. Templates folder and app.py should be there in the main folder which you have created.

STEP-1:

Create a GitHub Repository

- Log in to GitHub.
- Create a repository named **Sinzz0.github.io**.
- Do initialize with README, .gitignore , or license if needed.
- Copy the repository URL.

STEP-2:

Clone the Repository

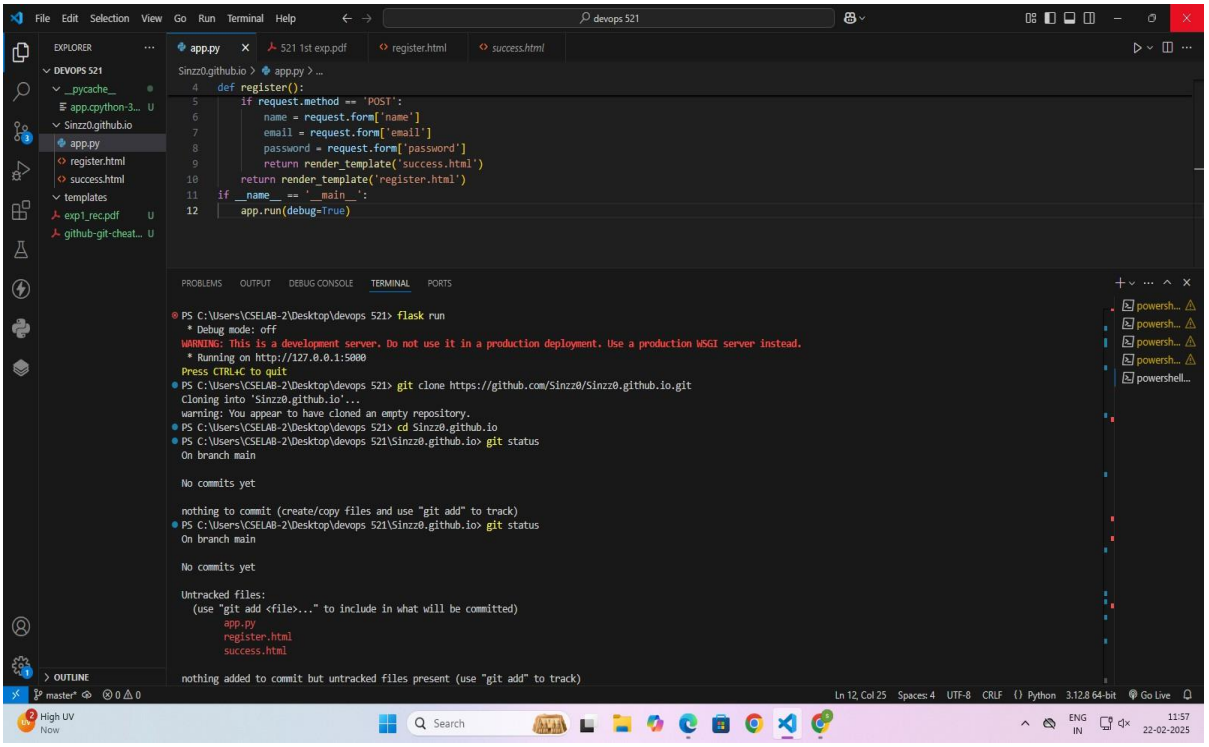
- Open VS Code and its terminal.
- Run the following : cd path\to\your\directory git clone <repository-url>
- git clone https://github.com/Sinzz0/Sinzz0.github.io.git

STEP-3:

Add Files and Folders

- Navigate to cloned repository:

  cd **Sinzz0.github.io**



After adding cd file, in the left side folders section, you will get a new folder named with your github repository. Now drag all the 3 files from experiment 1 into your newly created folder.

You can check the status by using git status.

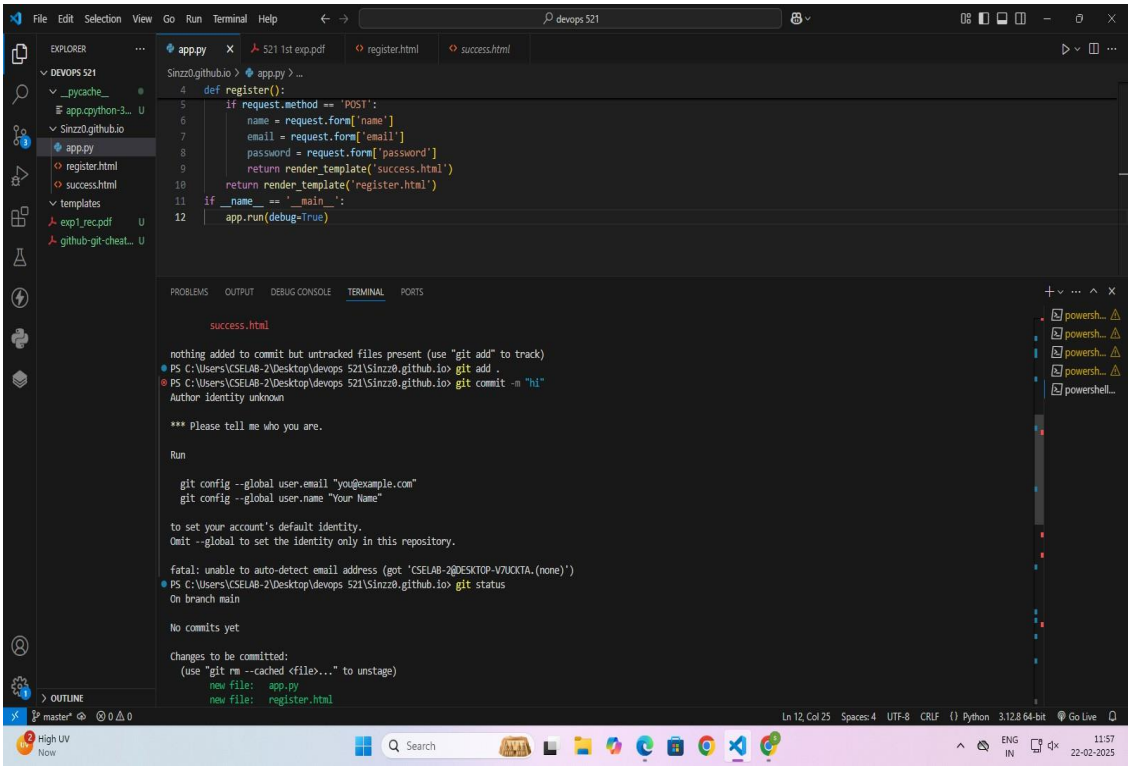
STEP-4:

Add, Commit, and Push Changes

Now do the command: git add

And next do the command: git commit –m “hi”

Then follow the instructions given and run the commands which were given in the terminal after running your previous command.



After running the commands given in terminal, again run:

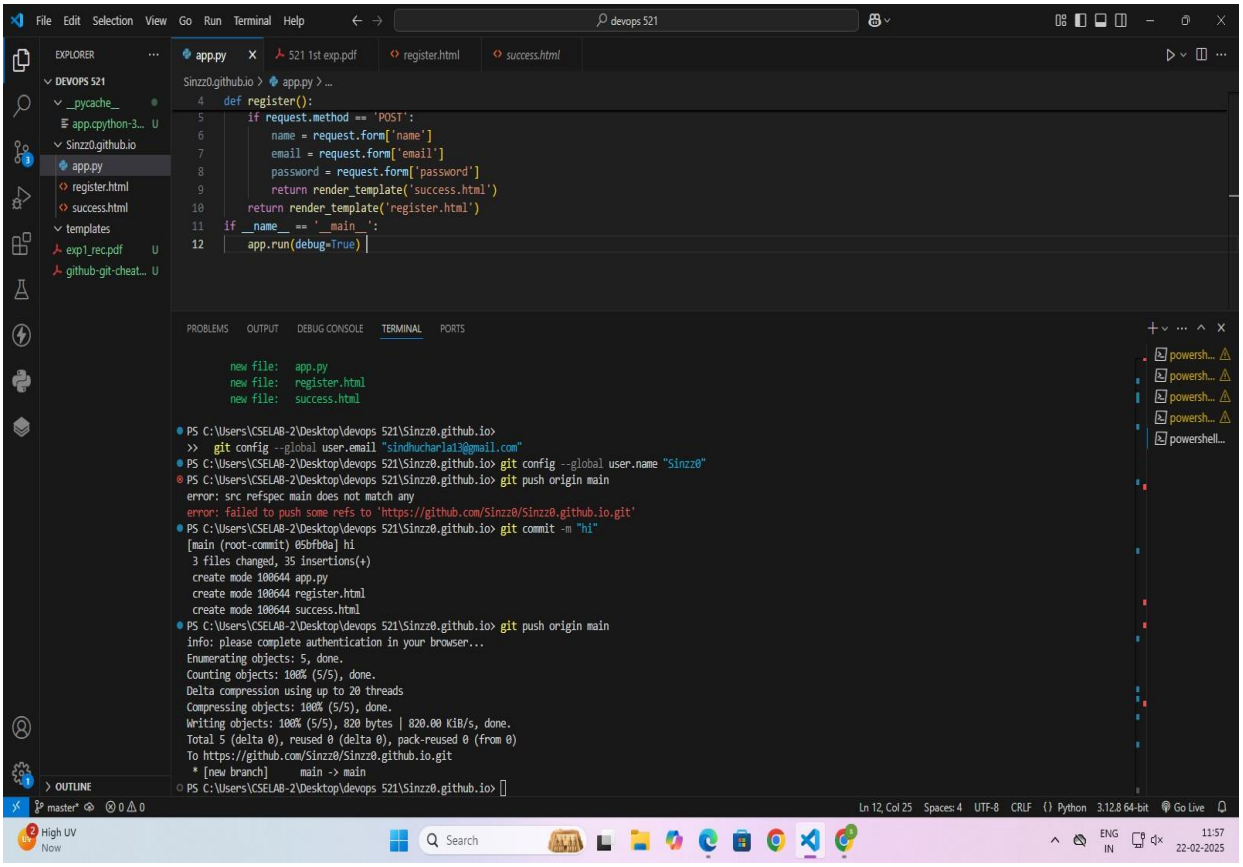
git commit –m “hi”

Now,

Push the files into your repository by using:

git push origin main

All the files will be pushed into your repository by using the above git commands.

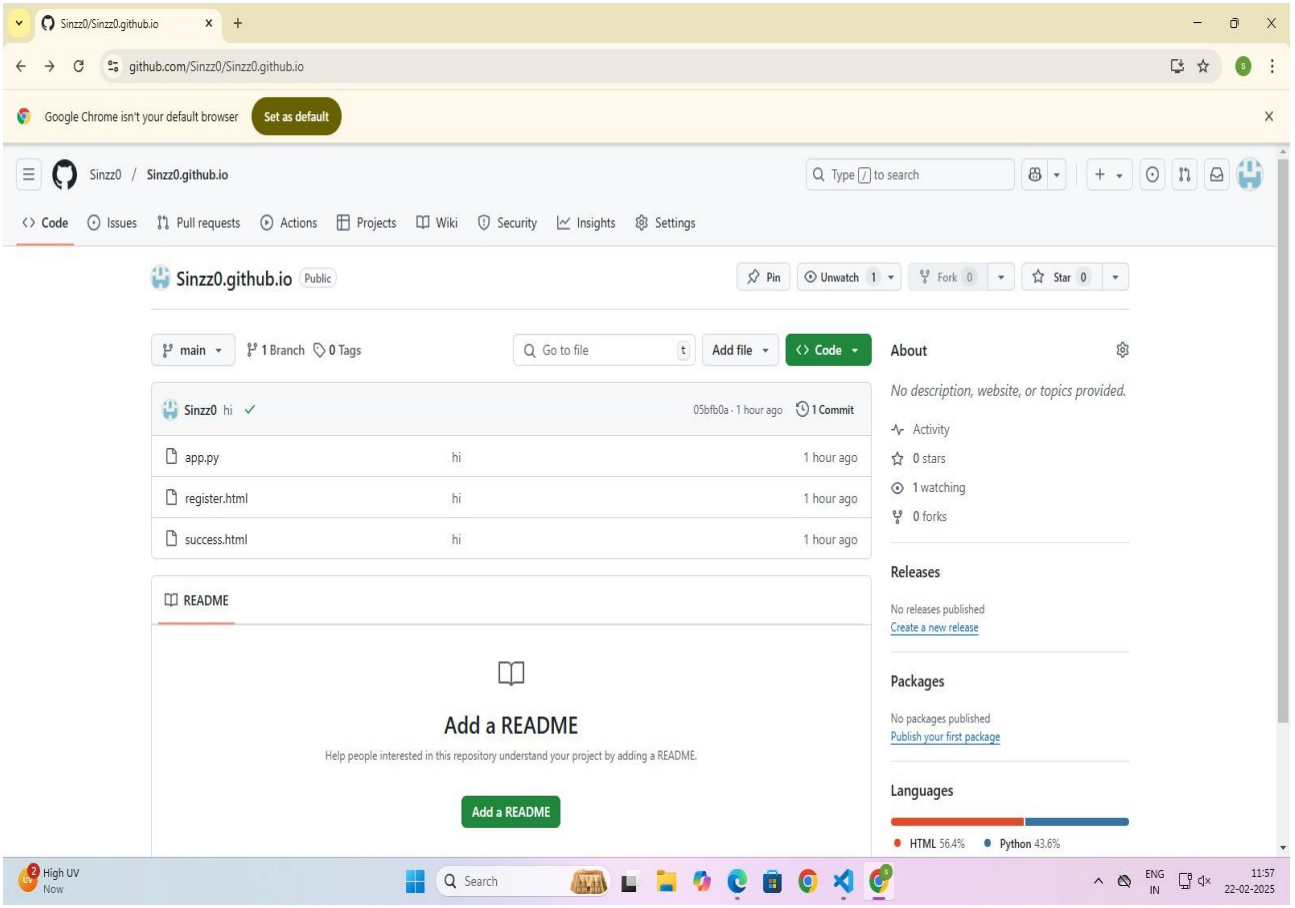


STEP-5:

Verify on GitHub

Go back to your repository on GitHub.

Now, refresh the page to ensure the app.py file and other 2 files are visible.


Sinzz0/Sinzz0.github.io
github.com/Sinzz0/Sinzz0.github.io
Google Chrome isn't your default browser
Set as default
Sinzz0 / Sinzz0.github.io
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Sinzz0.github.io
Public
Pin
Unwatch 1
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
Add file
Code
About
No description, website, or topics provided.
Sinzz0 hi
05bfb0a · 1 hour ago
1 Commit
app.py hi 1 hour ago
register.html hi 1 hour ago
success.html hi 1 hour ago
Activity
0 stars
1 watching
0 forks
README
Releases
No releases published
Create a new release
Add a README
Help people interested in this repository understand your project by adding a README.
Add a README
Packages
No packages published
Publish your first package
Languages
HTML 56.4%
Python 43.6%
High UV
Now
Search
ENG
IN
11:57
22-02-2025

EXPERIMENT NO: 4 **DATE:**

Aim:- Jenkins installation and setup, explore the environment.

Description:

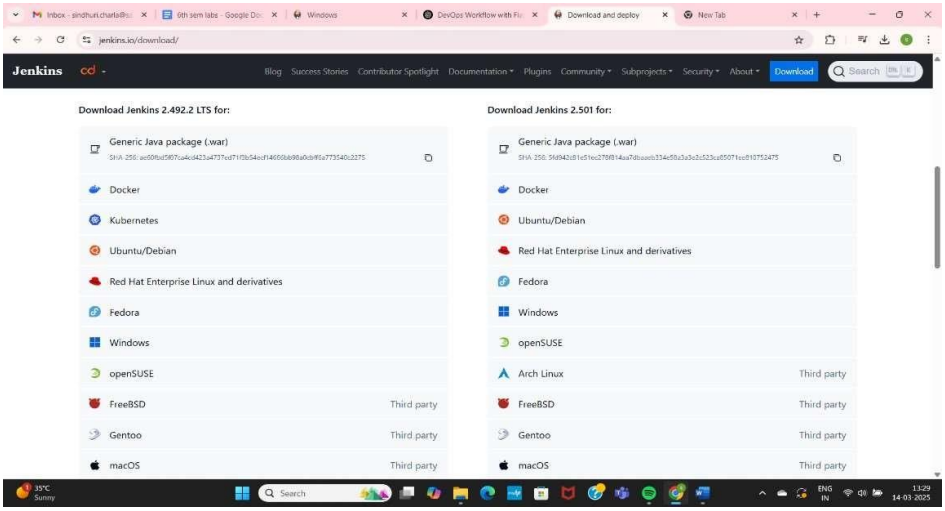
Installing Jenkins on Windows and configure it as a service. Explore its web-based interface to manage automation tasks and CI/CD pipelines.

Procedure:

First, Java should be installed in your systems. Version 17 or 21 are needed. Next,

STEP-1:

Install Jenkins for Windows

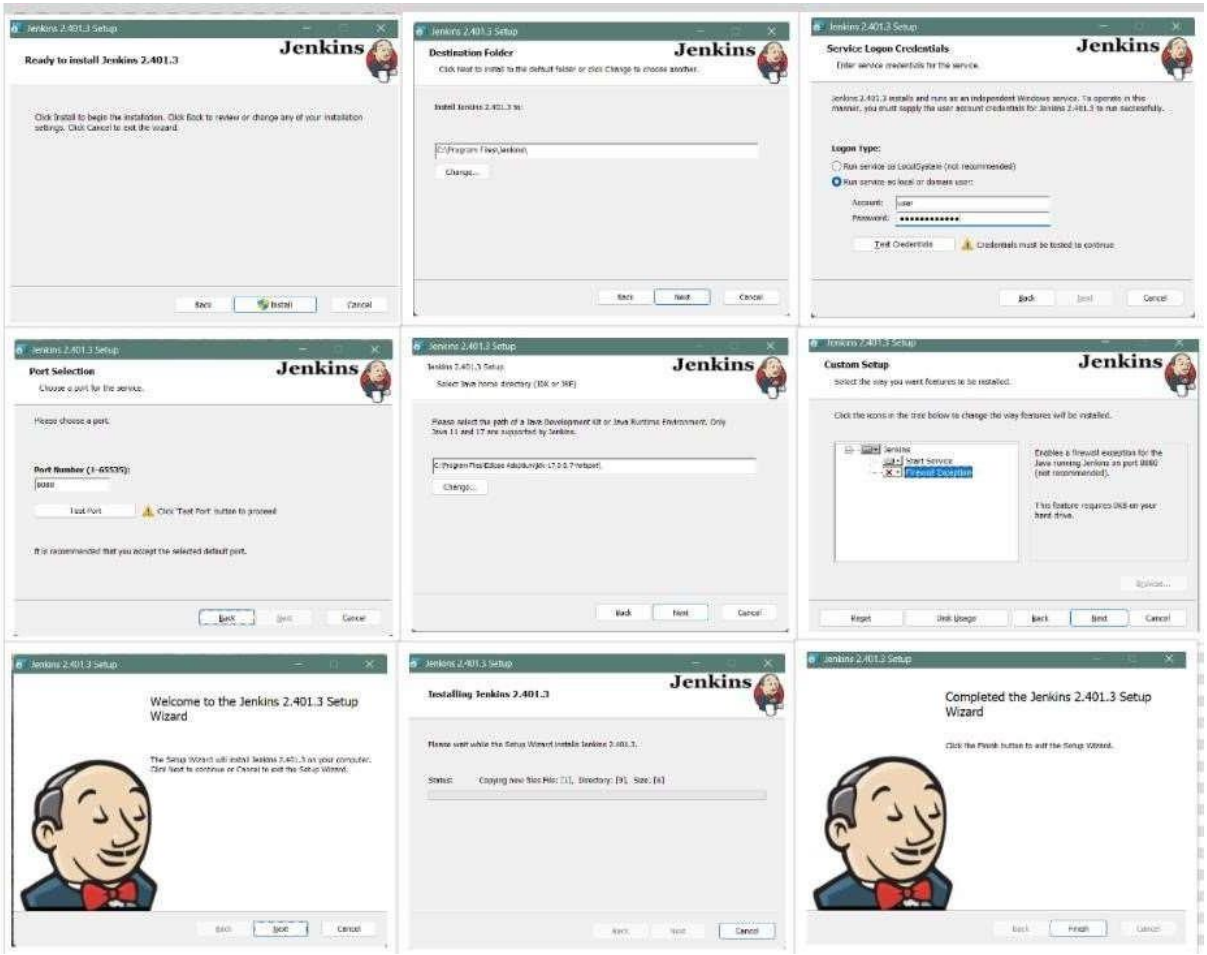


Download the Jenkins LTS file which you'll see on the left side. Click on the windows icon to begin the download.

STEP-2:

Next, complete all the steps as shown in the below images. Click next, after every step.

In 3rd step, give the username as 'admin' and password as 'password, only. After all the steps, Jenkins would be successfully installed !



After successfully installing, we need to set up the wizard.

Post-installation setup wizard

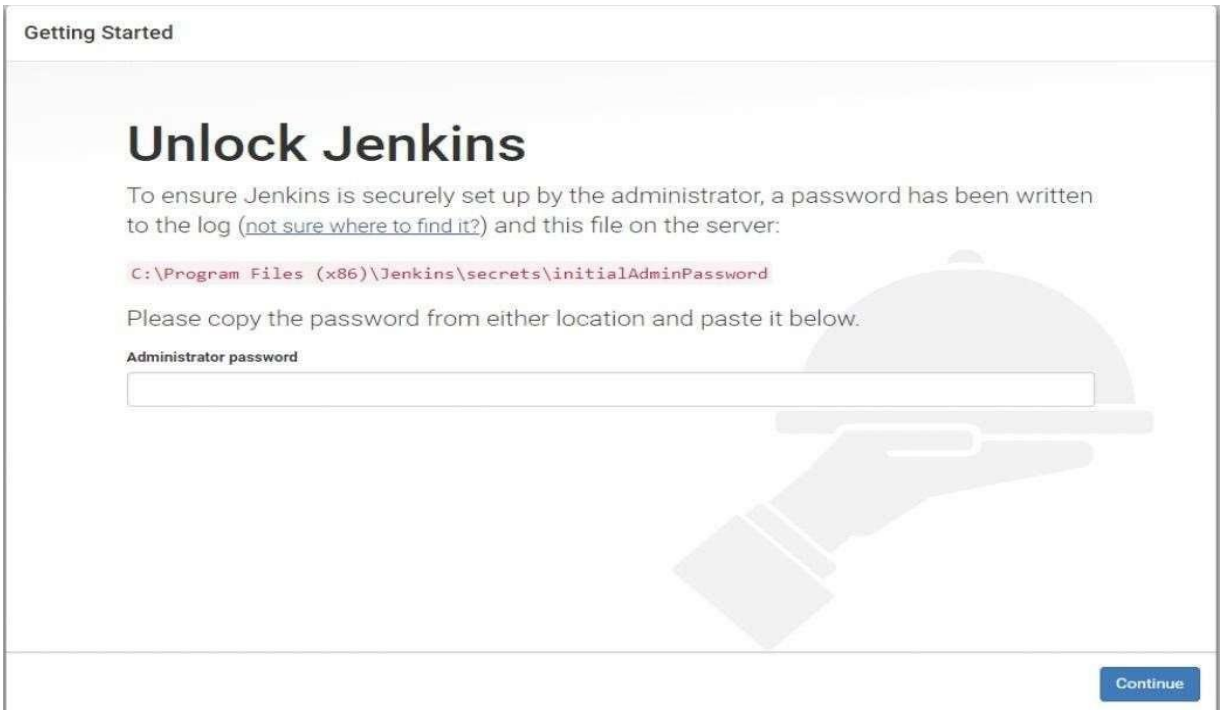
After downloading, installing, and running Jenkins, the post-installation setup wizard begins.

Unlocking Jenkins

You are asked to unlock a new Jenkins controller using an automatically generated password when you first access it.

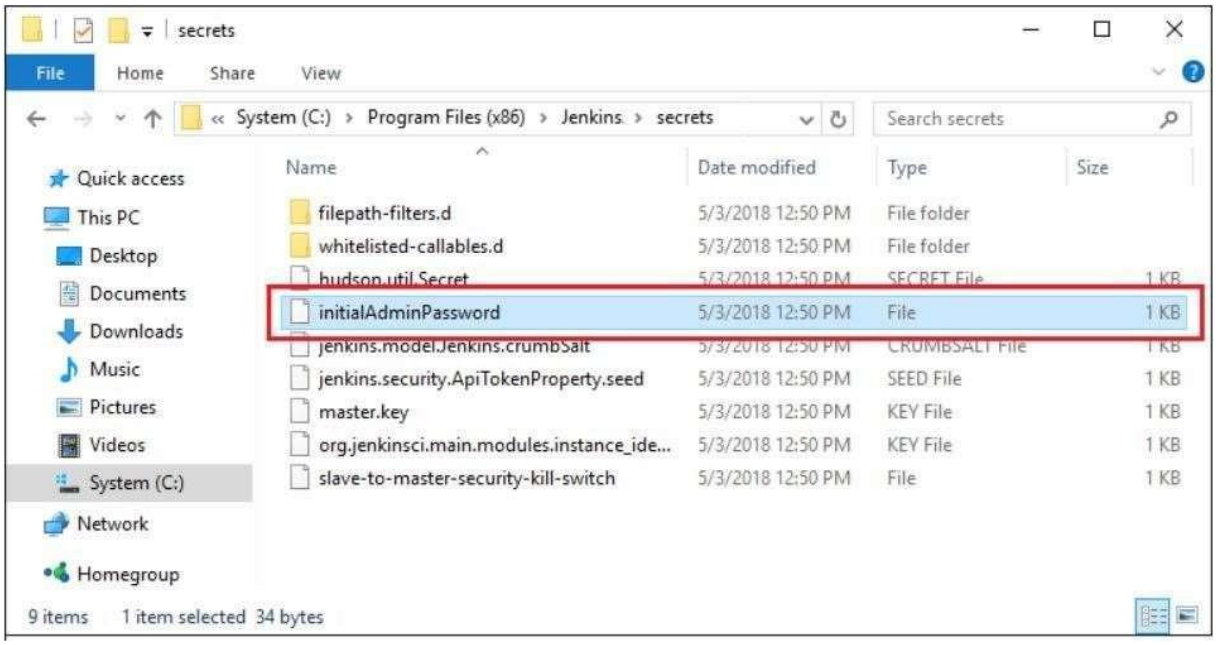
Step 1

Browse to http://localhost:8080



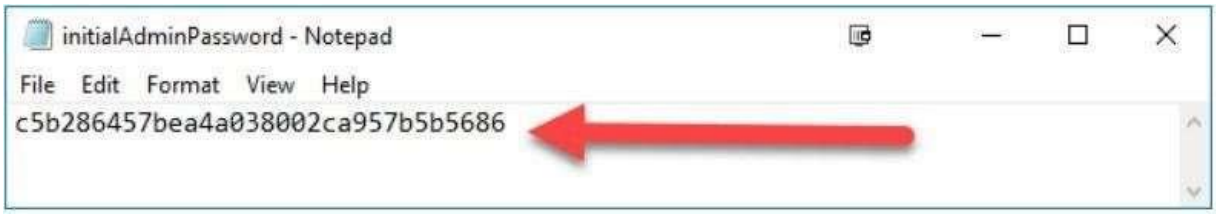
Step 2

The initial Administrator password should be found under the Jenkins installation path.



Step 3

Open the highlighted file and copy the content of the **initialAdminPassword** file.



Step 4

On the **Unlock Jenkins** page, paste this password into the **Administrator password** field and click **Continue**.

Customizing Jenkins with plugins

After unlocking Jenkins, the **Customize Jenkins** page appears. Here you can install any number of useful plugins as part of your initial setup.

Click one of the two options shown:

- **Install suggested plugins** - to install the recommended set of plugins, which are based on most common use cases.
- **Select plugins to install** - to choose which set of plugins to initially install. When you first access the plugin selection page, the suggested plugins are selected by default.

This process may take a few minutes.

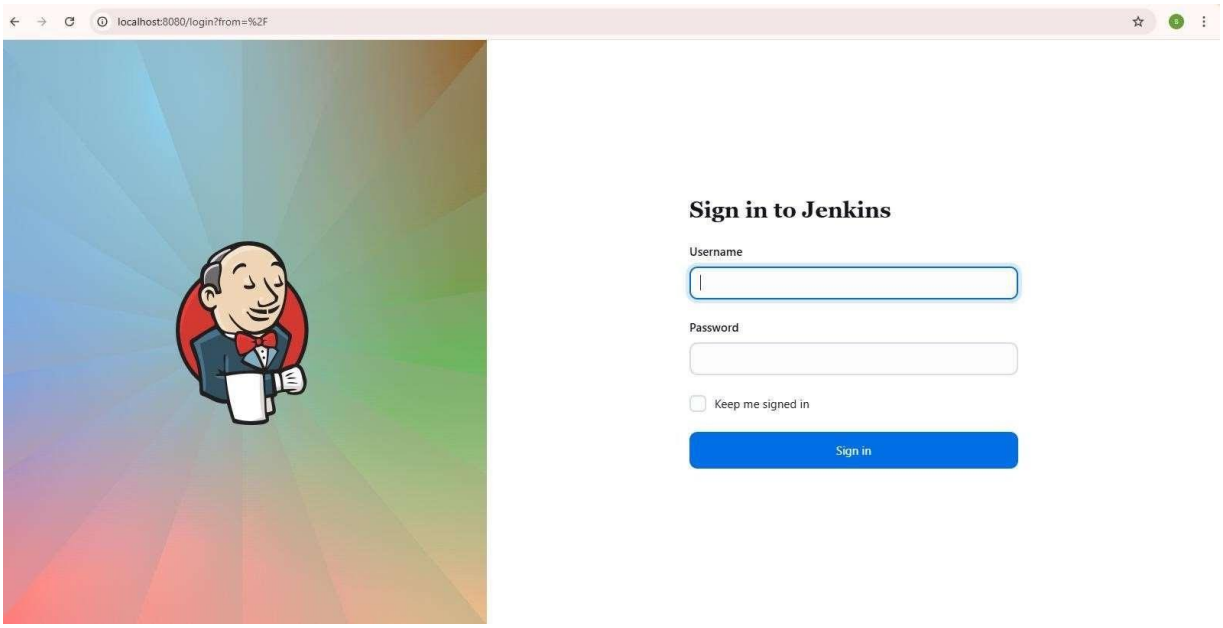
Creating the first administrator user

Finally, after customizing Jenkins with plugins, Jenkins asks you to create your first administrator user.

1. When the **Create First Admin User** page appears, specify the details for your administrator user in the respective fields and click **Save and Finish**.
2. Here, give your name and email if needed.
3. When the **Jenkins is ready** page appears, click **Start using Jenkins**.
4. If required, login to Jenkins with the credentials of the user you just created and you are ready to start using Jenkins!

After these things, Jenkins would be ready to use.
Open localhost:8080 and sign in to Jenkins.

It should look like this.



With this, we completed Jenkins installation and successfully explored its environment.

EXPERIMENT NO: 5 DATE:

Aim:- Demonstrate continuous integration and development using Jenkins.

Description:

Continuous Integration and Deployment using Jenkins: Configure Jenkins to automate the software development lifecycle, including building, testing, and deploying applications. Integrate

Procedure:

Step 1:

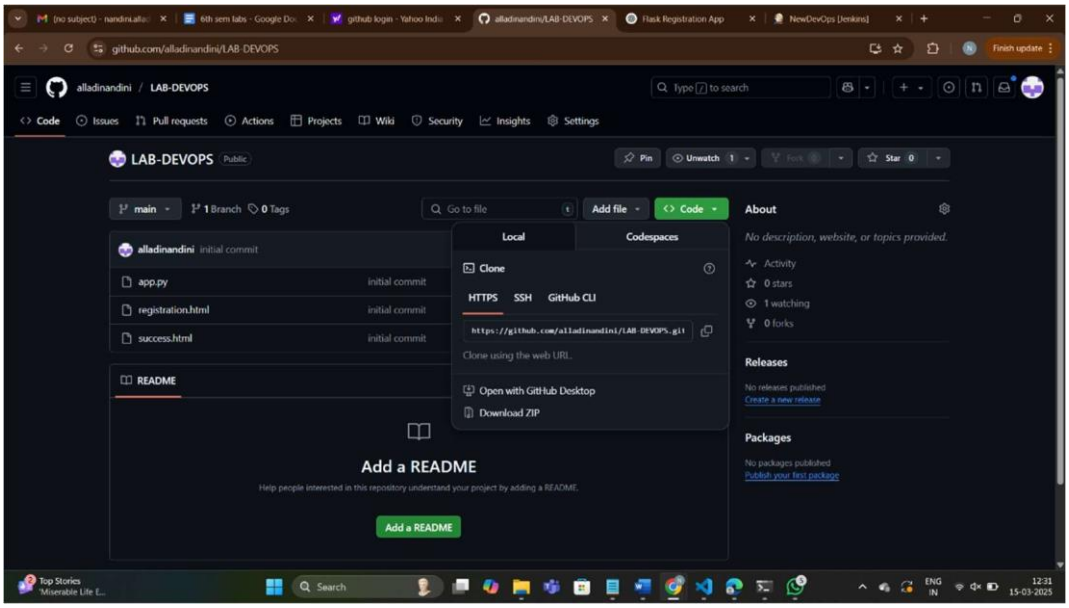
Create a New Jenkins Job

1. Open Jenkins Dashboard (http://localhost:8080).
2. Log in with your credentials.
3. Click on "New Item" in the left-hand menu.
4. Enter a name for your job (e.g., "DevOps, newDevOps, etc..") and select "Freestyle Project".
5. Click OK to create the job.

Step 2:

Configure GitHub as the Source Code Repository

1. Scroll down to the Source Code Management section.
2. Select Git (Ensure that Git is installed and accessible on your system).
3. In the Repository URL field, enter your GitHub repository URL
4. Copy the link which is shown in the below image.



Step 3:

Configure Build Triggers

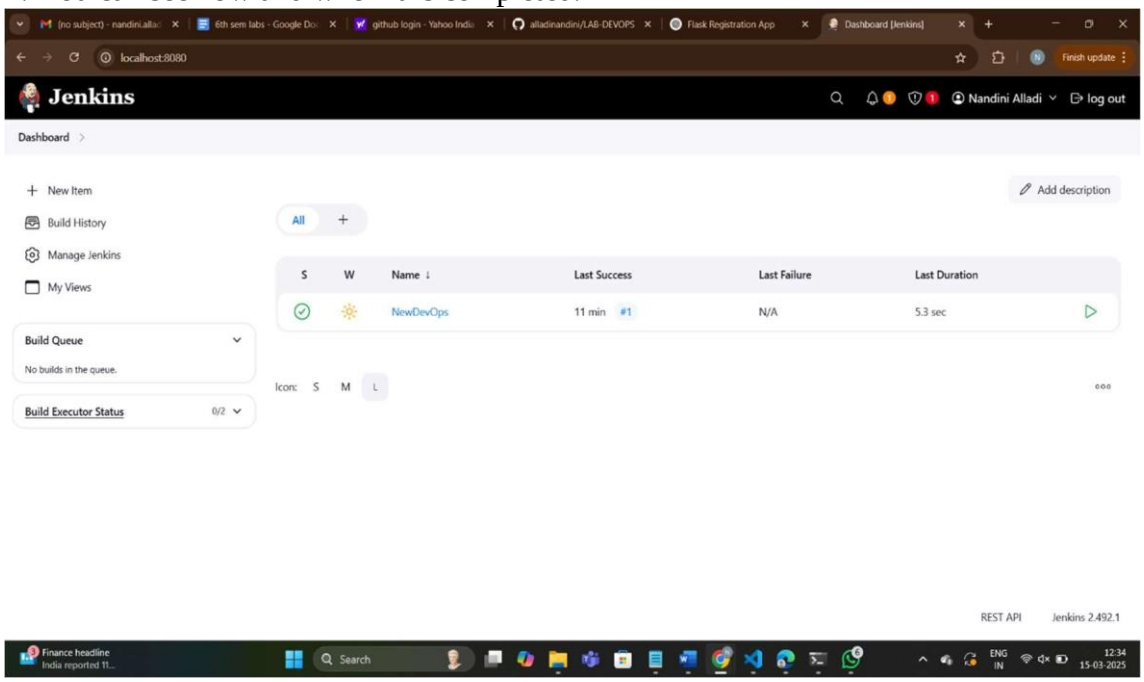
1. Scroll to the Build Triggers section.
2. Check the box "GitHub hook trigger for GITScm polling".
3. Click Apply to save changes.

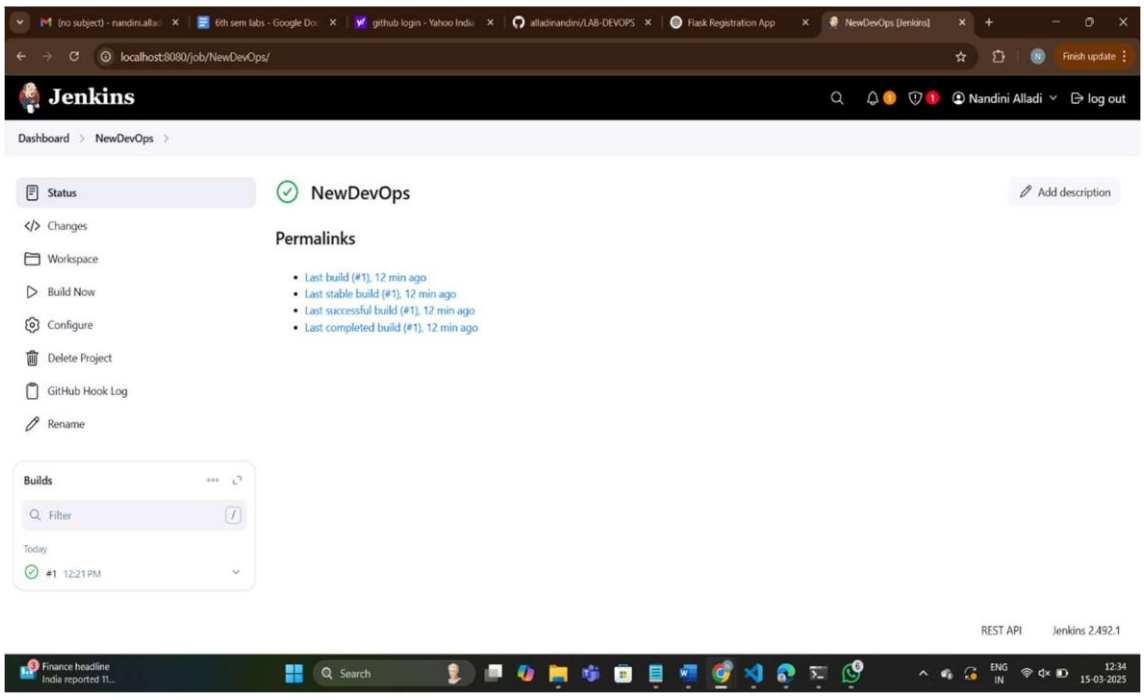
Step 4:

Save and Build the Job

1. Scroll to the bottom and click Save.
2. Click "Build Now" to manually tr

igger a build and check if everything works as expected.
3. Click on the newDevOps (#1, #2, etc.), to see the logs.
4. You can see how and when it is completed.





With this output, the experiment is completed.
We successfully demonstrated the usage of Jenkins.

EXPERIMENT NO:6 **DATE:**

Aim: Explore Docker commands for content management.

Description:

Docker is an **open-source containerization platform** that allows developers to build, package, and run applications in isolated environments called **containers**. It ensures that applications run **consistently across different computing environments**, eliminating compatibility issues.

Commands:

- **docker - -version**

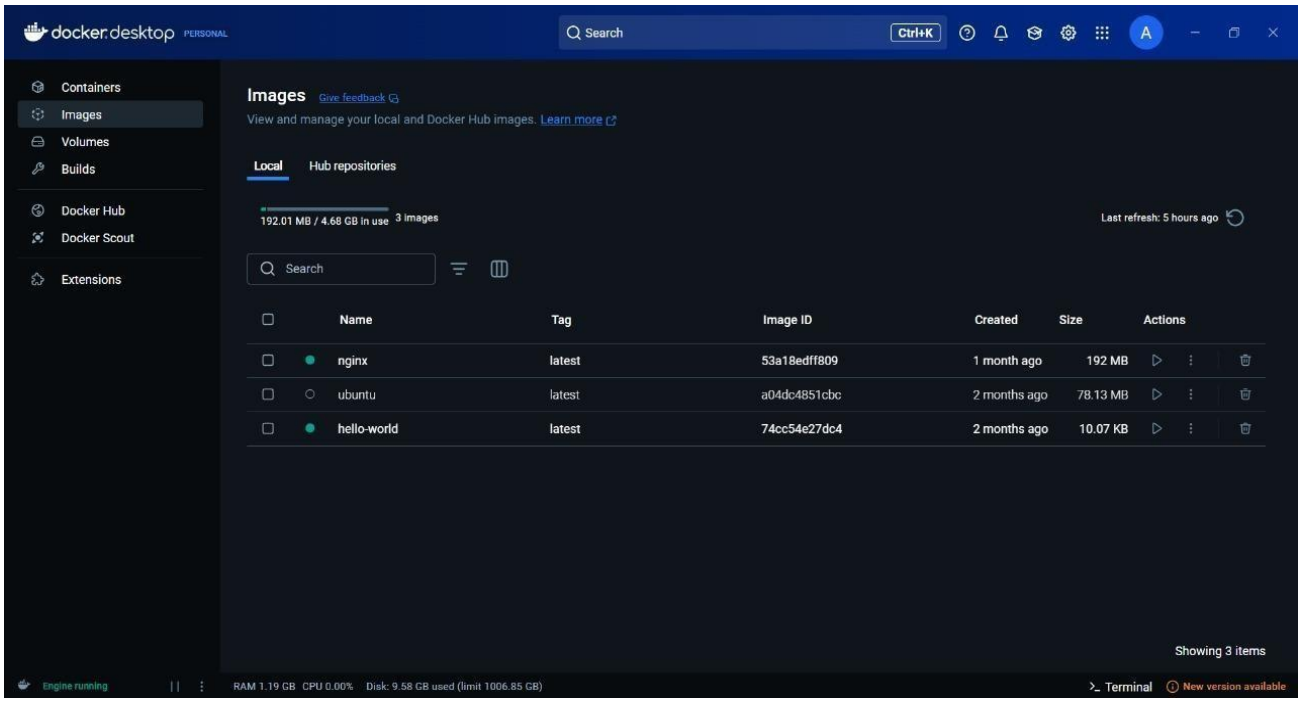
```
C:\Users\DELL>docker --version
Docker version 27.5.1, build 9f9e405

C:\Users\DELL>
```

- **docker pull hello-world**

```
C:\Users\DELL>docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
Digest: sha256:7e1a4e2d11e2ac7a8c3f768d4166c2defeb09d2a750b010412b6ea13de1efb19
Status: Image is up to date for hello-world:latest
docker.io/library/hello-world:latest
```

- **docker run hello-world**

```
C:\Users\DELL>docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```



- **docker run -d -p 8080:80 nginx**

```
C:\Users\DELL>docker run -d -p 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
6e909acdb790: Already exists
5eaa34f5b9c2: Pull complete
417c4bccf534: Pull complete
e7e0ca015e55: Pull complete
373fe654e984: Pull complete
97f5c0f51d43: Pull complete
c22eb46e871a: Pull complete
Digest: sha256:124b44bfc9ccd1f3cedf4b592d4d1e8bddb78b51ec2ed5056c52d3692baebc19
Status: Downloaded newer image for nginx:latest
23e7d0f5d6cee3bd303d1a32d20658aabb71fc60f5f95394ea3e0bbe6aaff853
```

- **docker pull ubuntu**

```
C:\Users\DELL>docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
5a7813e071bf: Already exists
Digest: sha256:72297848456d5d37d1262630108ab308d3e9ec7ed1c3286a32fe09856619a782
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

- **docker images**

```
C:\Users\DELL>docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx         latest    53a18edff809   6 weeks ago    192MB
ubuntu        latest    a04dc4851cbc   7 weeks ago    78.1MB
hello-world   latest    74cc54e27dc4   8 weeks ago    10.1kB
```

- **docker rmi -f hello-world**

```
C:\Users\DELL>docker rmi -f hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:7e1a4e2d11e2ac7a8c3f768d4166c2defeb09d2a750b010412b6ea13de1efb19
Deleted: sha256:74cc54e27dc41bb10dc4b2226072d469509f2f22f1a3ce74f4a59661a1d44602
```

- **docker ps**

```
C:\Users\DELL>docker ps
CONTAINER ID   IMAGE              COMMAND          CREATED       STATUS       PORTS                    NAMES
2849e2b03230   abinayateja/myapp   "python app.py"   2 hours ago   Up 2 hours   0.0.0.0:5000->5000/tcp   objective_bassi
```

- **docker ps -a**

```
C:\Users\DELL>docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS    PORTS     NAMES
23e7d0f5d6ce   nginx     "/docker-entrypoint.…"   26 minutes ago   Created             xenodochial_galois
```

- **docker image ls**

```
C:\Users\DELL>docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED       SIZE
flask-app           latest    13848130b025   4 hours ago   145MB
myapp               latest    25d83f3d1ab7   9 hours ago   134MB
abinayateja/myapp   latest    25d83f3d1ab7   9 hours ago   134MB
nginx               latest    53a18edff809   6 weeks ago   192MB
ubuntu              latest    a04dc4851cbc   7 weeks ago   78.1MB
```

- **docker ps -a**

```
C:\Users\DELL>docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS    PORTS     NAMES
23e7d0f5d6ce   nginx     "/docker-entrypoint.…"   26 minutes ago   Created             xenodochial_galois
```

- **docker inspect image nginx**

```
C:\Users\DELL>docker inspect image nginx
[
    {
        "Id": "sha256:53a18edff8091d5faff1e42b4d885bc5f0f897873b0b8f0ace236cd5930819b0",
        "RepoTags": [
            "nginx:latest"
        ],
        "RepoDigests": [
            "nginx@sha256:124b44bfc9ccd1f3cedf4b592d4d1e8bddb78b51ec2ed5056c52d3692baebc19"
        ],
        "Parent": "",
        "Comment": "buildkit.dockerfile.v0",
        "Created": "2025-02-05T21:27:16Z",
        "DockerVersion": "",
        "Author": "",
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "NGINX_VERSION=1.27.4",
                "NJS_VERSION=0.8.9",
                "NJS_RELEASE=1~bookworm",
                "PKG_RELEASE=1~bookworm",
                "DYNPKG_RELEASE=1~bookworm"
            ],
            "Cmd": [
                "nginx",
                "-g",
                "daemon off;"
            ],
            "Image": "",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": [
                "/docker-entrypoint.sh"
            ],
            "OnBuild": null,
            "Labels": {
                "maintainer": "NGINX Docker Maintainers <docker-maint@nginx.com>"
            },
```

- **docker restart a2a8ef430f9b**

```
C:\Users\DELL>docker restart 23e7d0f5d6ce
23e7d0f5d6ce
```

- **docker stop xenodochial_galois**

```
C:\Users\DELL>docker stop  xenodochial_galois
xenodochial_galois

C:\Users\DELL>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

- **docker run -d -p 9090:8080 - -name my_container nginx**

```
C:\Users\DELL>docker run -d -p 9090:8080 --name my_container nginx
6660950227d2281de2e7449d646ff954bb8050d6579456f3e32f78af17fc6576

C:\Users\DELL>docker ps -a
CONTAINER ID   IMAGE            COMMAND              CREATED         STATUS                     PORTS                          NAMES
6660950227d2   nginx            "/docker-entrypoint.…"   11 seconds ago   Up 9 seconds               80/tcp, 0.0.0.0:9090->8080/tcp   my_container
2849e2b03230   abinayateja/myapp   "python app.py"      2 hours ago     Up 2 hours                 0.0.0.0:5000->5000/tcp         objective_bassi
1ad674584d7c   myapp            "python app.py"      2 hours ago     Exited (0) 2 hours ago                                    lucid_knuth
cf164c3da907   myapp            "python app.py"      2 hours ago     Created                                                   quirky_colden
b26c6888adb3   myapp            "python app.py"      2 hours ago     Created                                                   hardcore_babbage
```

- **docker rm -f my_container**

```
C:\Users\DELL>docker rm -f my_container
my_container

C:\Users\DELL>docker ps -a
CONTAINER ID   IMAGE              COMMAND          CREATED       STATUS                     PORTS                    NAMES
2849e2b03230   abinayateja/myapp  "python app.py"  2 hours ago   Up 2 hours                 0.0.0.0:5000->5000/tcp   objective_bassi
1ad674584d7c   myapp              "python app.py"  2 hours ago   Exited (0) 2 hours ago                              lucid_knuth
cf164c3da907   myapp              "python app.py"  2 hours ago   Created                                             quirky_colden
b26c6888adb3   myapp              "python app.py"  2 hours ago   Created                                             hardcore_babbage
```

- **docker build -t flask-app .**

```
PS C:\Users\DELL\OneDrive\Desktop\DevOpsMain\DevOpsLab> docker build -t flask-app .
[+] Building 4.4s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 526B
 => [internal] load metadata for docker.io/library/python:3.11-slim
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/library/python:3.11-slim@sha256:7029b00486ac40bed03e36775b864d3f3d39dcbdf19cd45e6a52d541e6c178f0
 => [internal] load build context
 => => transferring context: 3.99kB
 => CACHED [2/5] WORKDIR /app
 => CACHED [3/5] COPY requirements.txt .
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt
 => [5/5] COPY . .
 => exporting to image
 => => exporting layers
 => => writing image sha256:13848130b025c1f777525ff6e85af9f15043f21b757d08175fac0d1cec85ebcb
 => => naming to docker.io/library/flask-app
```

- **docker run -p 5000:5000 flask-app**

```
PS C:\Users\DELL\OneDrive\Desktop\DevOpsMain\DevOpsLab> docker run -p 5000:5000 flask-app
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 105-272-758
172.17.0.1 - - [21/Mar/2025 07:24:51] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [21/Mar/2025 07:25:00] "POST / HTTP/1.1" 200 -
```

EXPERIMENT NO:7 DATE:

Aim: Develop a simple containerized application using docker.

Description: This is a simple containerized Flask application using Docker. It serves a basic web page displaying "Hello, Dockerized World!" on port 5000. The app is built using a Python 3.9 base image and dependencies are managed via requirements.txt. The Docker image can be pushed to Docker Hub for easy deployment. Users can run it with a single docker run command.

Step-1: Install Docker

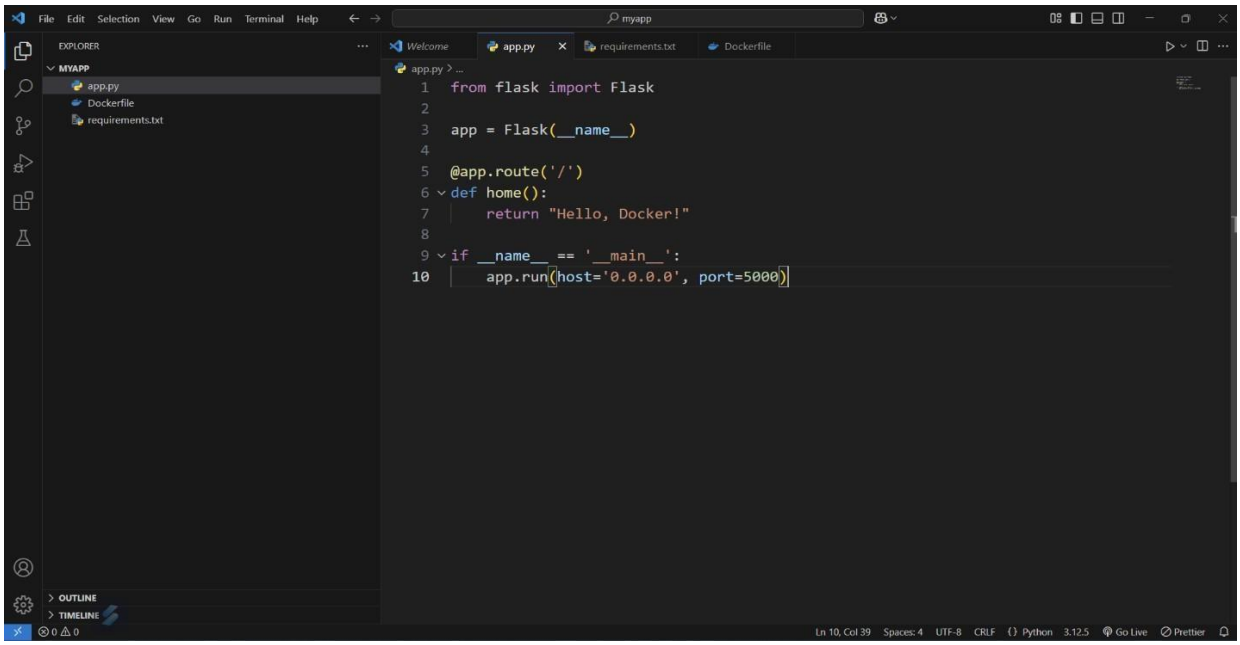
- Download and install Docker from Docker's official site.
- Verify installation
- **docker –version**

Step-2: Create a Project Directory

- **Create a folder** myapp and open it in command prompt
- mkdir myapp && cd myapp
- And also same folder open in vs code also.

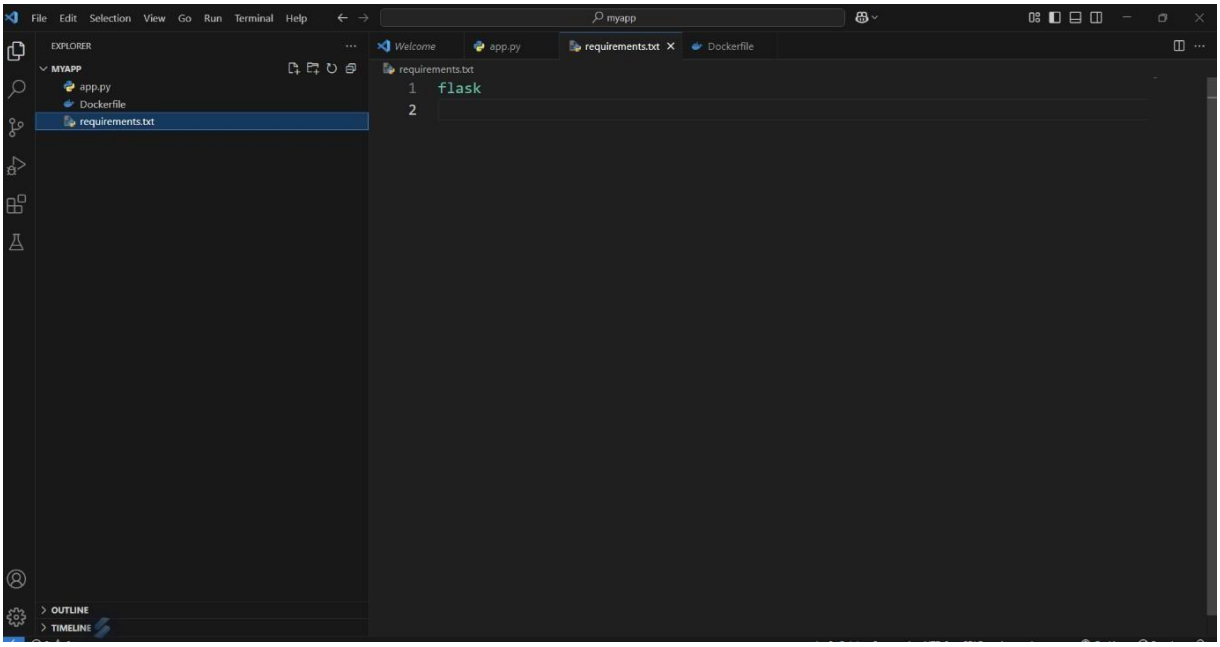
Step-3: Create Application Code.

- Create **app.py** in vs code and write the simple python flask application.

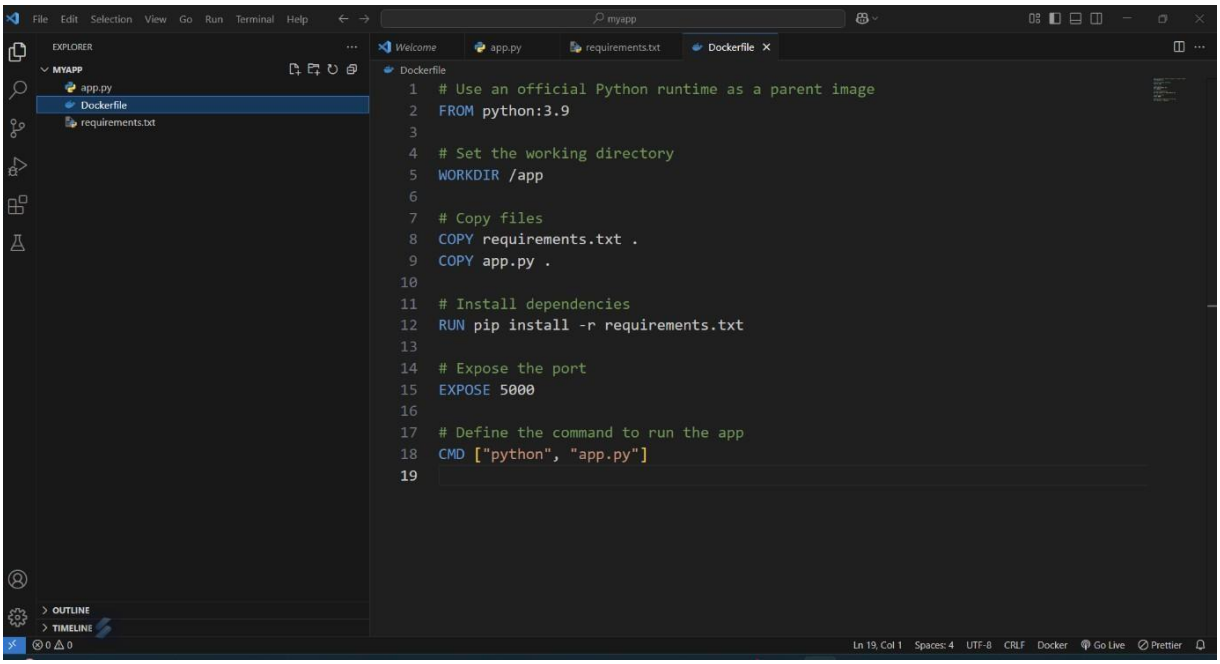


Step-4: Create requirements.txt

- And paste the **flask** in this file.



Step-5: Create a **Dockerfile** in vs code.



Step-6: Build Docker Image in command prompt.

- docker build -t myapp .

```
C:\Users\legal\OneDrive\Documents\myapp>docker build -t myapp .
[+] Building 11.9s (11/11) FINISHED                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                   0.1s
 => => transferring dockerfile: 371B                                                                                   0.0s
 => [internal] load metadata for docker.io/library/python:3.9                                                          3.6s
 => [auth] library/python:pull token for registry-1.docker.io                                                          0.0s
 => [internal] load .dockerignore                                                                                      0.0s
 => => transferring context: 2B                                                                                        0.0s
 => [1/5] FROM docker.io/library/python:3.9@sha256:5ea663a1c6ba266fdcac5949d1d2ea364ce30a2da92a3df95bb3c01437633a      0.0s
 => [internal] load build context                                                                                      0.1s
 => => transferring context: 268B                                                                                      0.0s
 => CACHED [2/5] WORKDIR /app                                                                                          0.0s
 => [3/5] COPY requirements.txt .                                                                                      0.0s
 => [4/5] COPY app.py .                                                                                                0.1s
 => [5/5] RUN pip install -r requirements.txt                                                                          7.5s
 => exporting to image                                                                                                 0.3s
 => => exporting layers                                                                                                0.2s
 => => writing image sha256:860dfcf82ff8d9f7c64b18b332dcca9e0c5c31599aa9ec3acabcf99a1fb681cf                           0.0s
 => => naming to docker.io/library/myapp                                                                               0.0s
```

Step-7: Run the Docker Container

- docker run -p 5000:5000 myapp

```
C:\Users\legal\OneDrive\Documents\myapp>docker run -p 5000:5000 myapp
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [16/Mar/2025 11:34:06] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [16/Mar/2025 11:34:08] "GET /favicon.ico HTTP/1.1" 404 -
172.17.0.1 - - [16/Mar/2025 11:38:41] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [16/Mar/2025 11:38:42] "GET /favicon.ico HTTP/1.1" 404 -
```

Step-8: Your app will be available at: http://localhost:5000

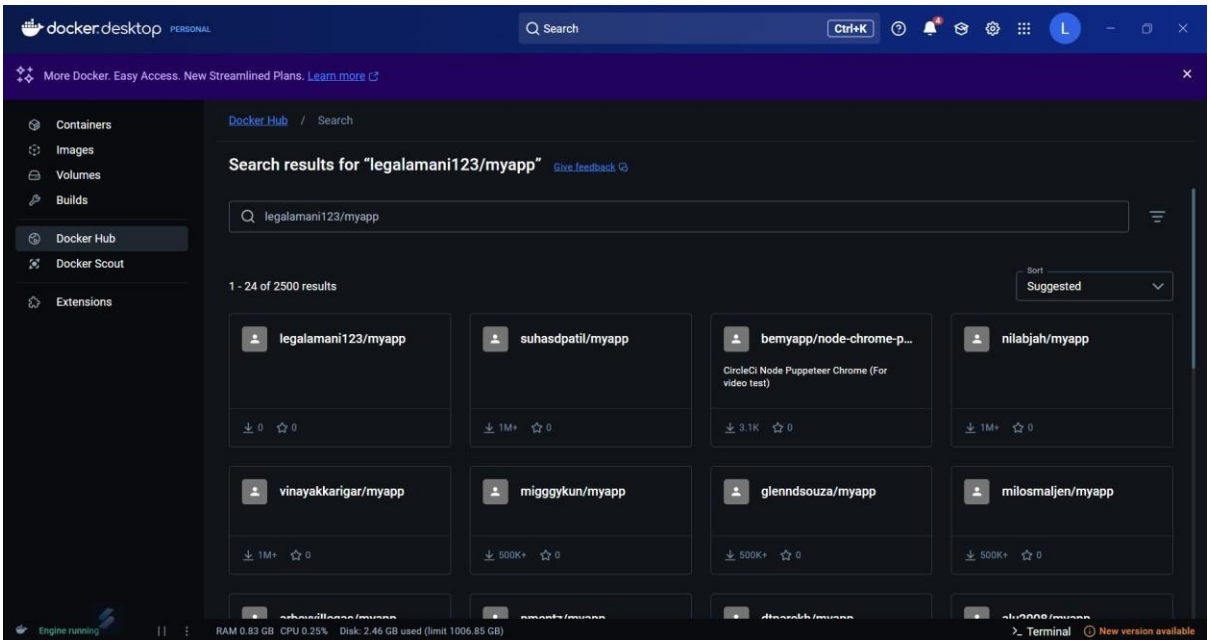
Step-9: Tag your built image to match your Docker Hub repository format**.**

- **docker tag myapp**
- Push the Image to Docker Hub.
- **docker push myapp**
- using these commands to push the docker hub

```
C:\Users\legal\OneDrive\Documents\myapp>docker tag myapp legalamani123/myapp

C:\Users\legal\OneDrive\Documents\myapp>docker push legalamani123/myapp
Using default tag: latest
The push refers to repository [docker.io/legalamani123/myapp]
8455839c94d2: Pushed
b31d55e7f923: Pushed
fe59ae57cf98: Pushed
1468b9ffa057: Pushed
01db3e67097a: Mounted from library/python
e49d0c94aa2a: Mounted from library/python
1c86760c5c93: Mounted from library/python
4b017a36fd9c: Mounted from library/python
20a9b386e10e: Mounted from library/python
f8217d7865d2: Mounted from library/python
01c9a2a5f237: Mounted from library/python
latest: digest: sha256:87a13ba97d8735b3866002b9d3d1c738ac1c79a56f3aa20a21b0b0748bb1291d size: 2627
```

Step-10: Verify the Image on Docker Hub

Go to Docker Hub and check if the image myapp appears under your account legalamani123.



Step-11: Run Your Image from Docker Hub (Anywhere)

- **docker run -p 5000:5000 myapp**

```
C:\Users\legal\OneDrive\Documents\myapp>docker run -p 5000:5000 legalamani123/myapp
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [16/Mar/2025 11:48:51] "GET / HTTP/1.1" 200 -
```

EXPERIMENT NO: 8 **DATE:**

Aim: Integrate kubernetes and docker.

Description: Integrating Kubernetes and Docker allows containerized applications to be managed at scale. First, enable Kubernetes in Docker Desktop. Build and push your Docker image to a registry. Create a Kubernetes Deployment using the image. Expose it via a Service (NodePort or LoadBalancer). Use kubectl port-forward or kubectl get services to access the app. Verify integration with kubectl get nodes.

Step-1: Install Docker and Enable Kubernetes

1. Install Docker Desktop from Docker's official site.
2. Open Docker Desktop, go to Settings > Kubernetes, and enable Kubernetes.
3. Wait for Kubernetes to start (verify with kubectl version).

Step-2: Build and Push Docker Image •

Docker build -t myapp .

```
C:\Users\legal\OneDrive\Documents\myapp>docker build -t myapp .
[+] Building 15.0s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 371B
 => [internal] load metadata for docker.io/library/python:3.9
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/library/python:3.9@sha256:5ea663a1c6ba266fdcac5949d1d2ea364ce30a2da92a3df95bb3c01437633ad9
 => [internal] load build context
 => => transferring context: 249B
 => CACHED [2/5] WORKDIR /app
 => CACHED [3/5] COPY requirements.txt .
 => [4/5] COPY app.py .
 => [5/5] RUN pip install -r requirements.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:545a762df1d72ef757c9475dee714f2e37c97865a59ce549bd8fbffd6387efce
 => => naming to docker.io/library/myapp
```

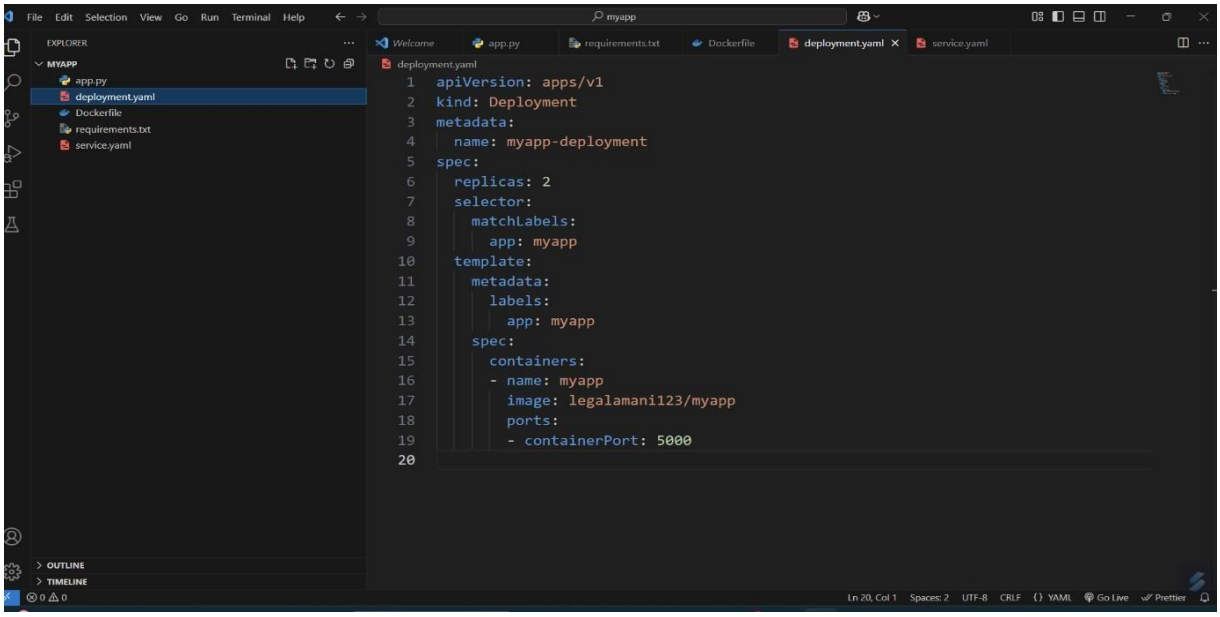
Step-3: run these commands

- docker tag myapp
- docker push myapp

```
C:\Users\legal\OneDrive\Documents\myapp>docker tag myapp legalamani123/myapp

C:\Users\legal\OneDrive\Documents\myapp>docker push legalamani123/myapp
Using default tag: latest
The push refers to repository [docker.io/legalamani123/myapp]
cc87a302a404: Pushed
8329570bf2fc: Pushed
fe59ae57cf98: Layer already exists
1468b9ffa057: Layer already exists
01db3e67097a: Layer already exists
e49d0c94aa2a: Layer already exists
1c86760c5c93: Layer already exists
4b017a36fd9c: Layer already exists
20a9b386e10e: Layer already exists
f8217d7865d2: Layer already exists
01c9a2a5f237: Layer already exists
latest: digest: sha256:164ab9bf36ecd4c9a00ff6d91ca3e9c716071296560af0233403ac62c9520e26 size: 2627
```

Step-4: Create Kubernetes Deployment.

Create a file deployment.yaml:

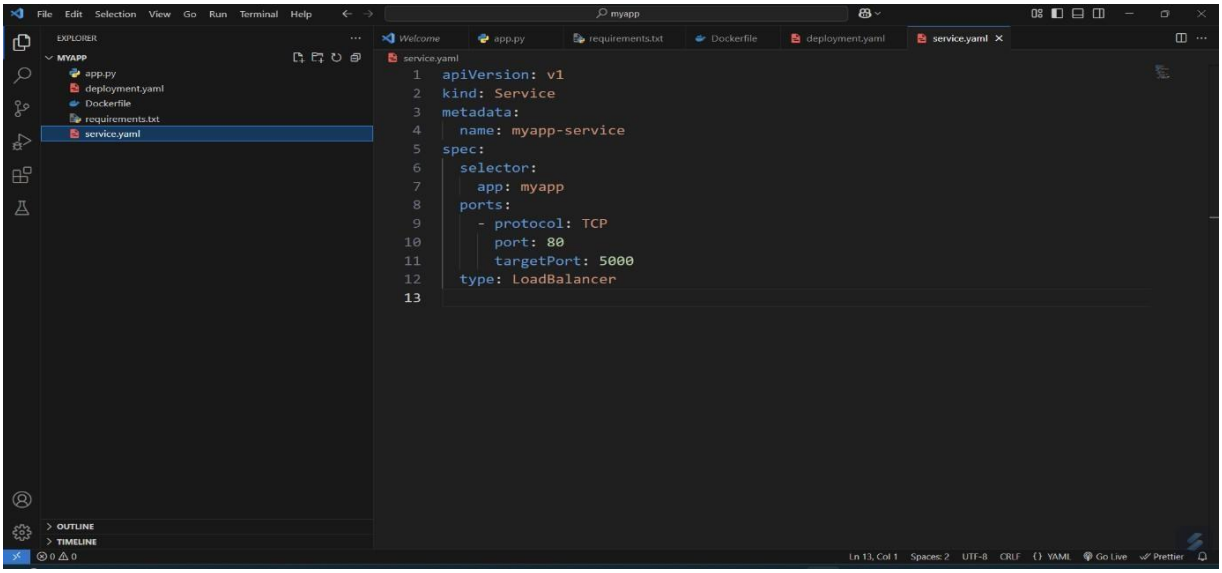


- Run in command prompt

```
C:\Users\legal\OneDrive\Documents\myapp>kubectl apply -f deployment.yaml
deployment.apps/myapp-deployment created
```

Step-5: Expose the Application

- Create a service.yaml file



- Run in command prompt in this command

```
C:\Users\legal\OneDrive\Documents\myapp>kubectl apply -f service.yaml
service/myapp-service created
```

Step-6: Access the Application

Find the service URL:

- **kubectl get services**

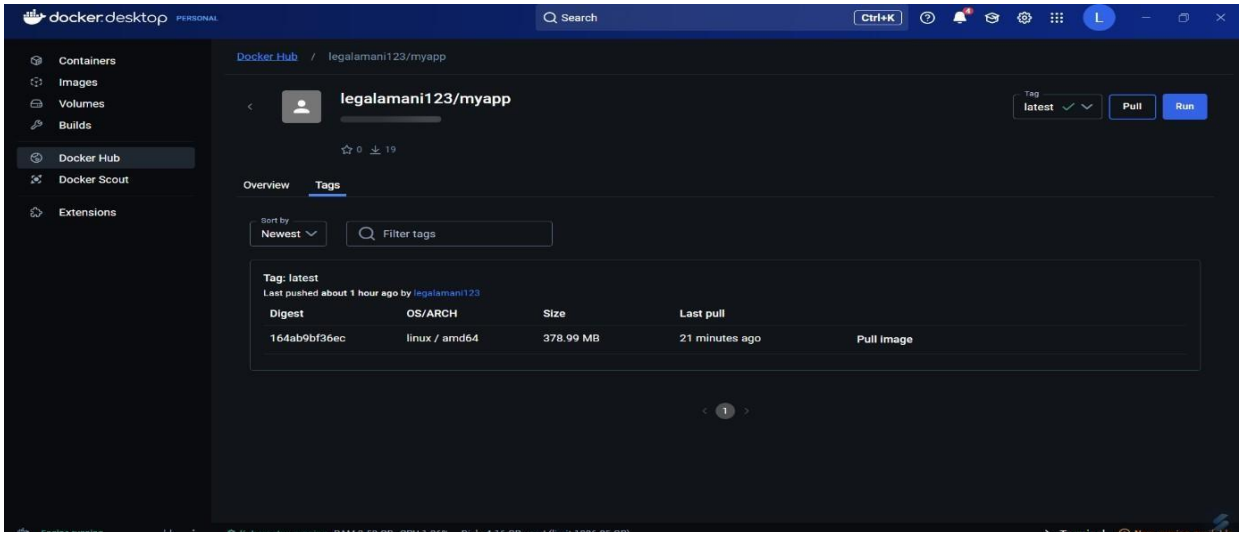
```
C:\Users\legal\OneDrive\Documents\myapp>kubectl get services
NAME            TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP      10.96.0.1      <none>        443/TCP        37m
myapp-service   LoadBalancer   10.109.53.7    localhost     80:31822/TCP   26m
```

- **kubectl get pods**

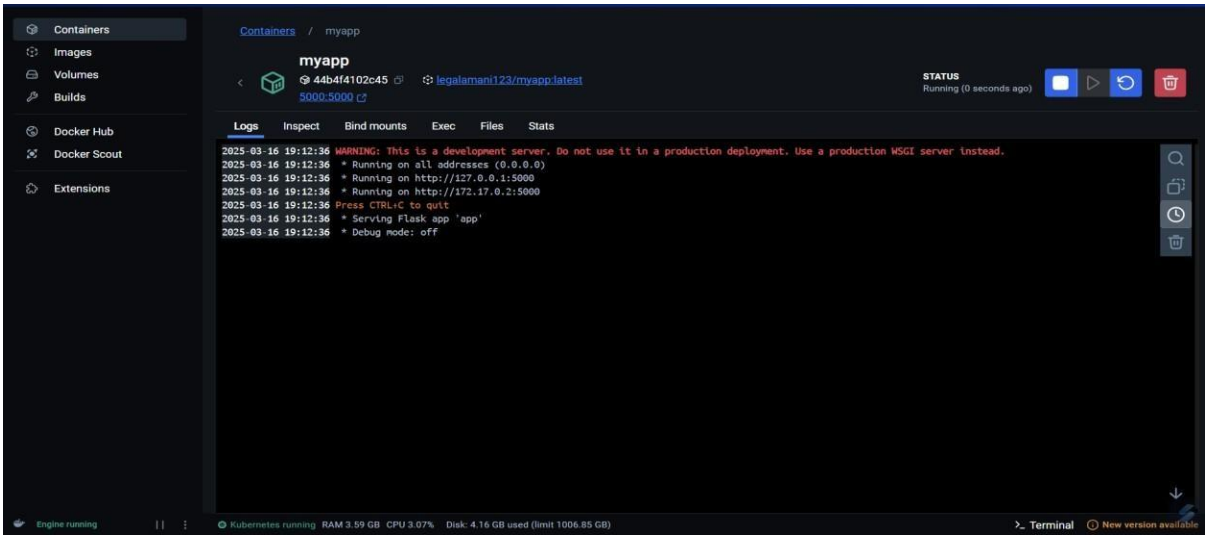
```
C:\Users\legal\OneDrive\Documents\myapp>kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
myapp-deployment-755b79564-2csjh     1/1     Running   0          15m
myapp-deployment-755b79564-pkvpt     1/1     Running   0          15m
```

Step-7: Now open http://localhost:5000 in your browser.

Step-8: After integrate kubernets and docker.It will be shown in docker hub and also run it.



- **After run button click it will run.**



- **Then click host services it will shows output.**

EXPERIMENT NO: 9 **DATE:**

Aim:

Automate the process of running a containerized application developed in **Exercise 7** using **Kubernetes**.

Description:

This experiment focuses on deploying a containerized application in **Kubernetes** on Windows using **Docker Desktop with Kubernetes** or **Minikube**. We will create Kubernetes **deployments and services**, apply them using kubectl, and verify the deployment.

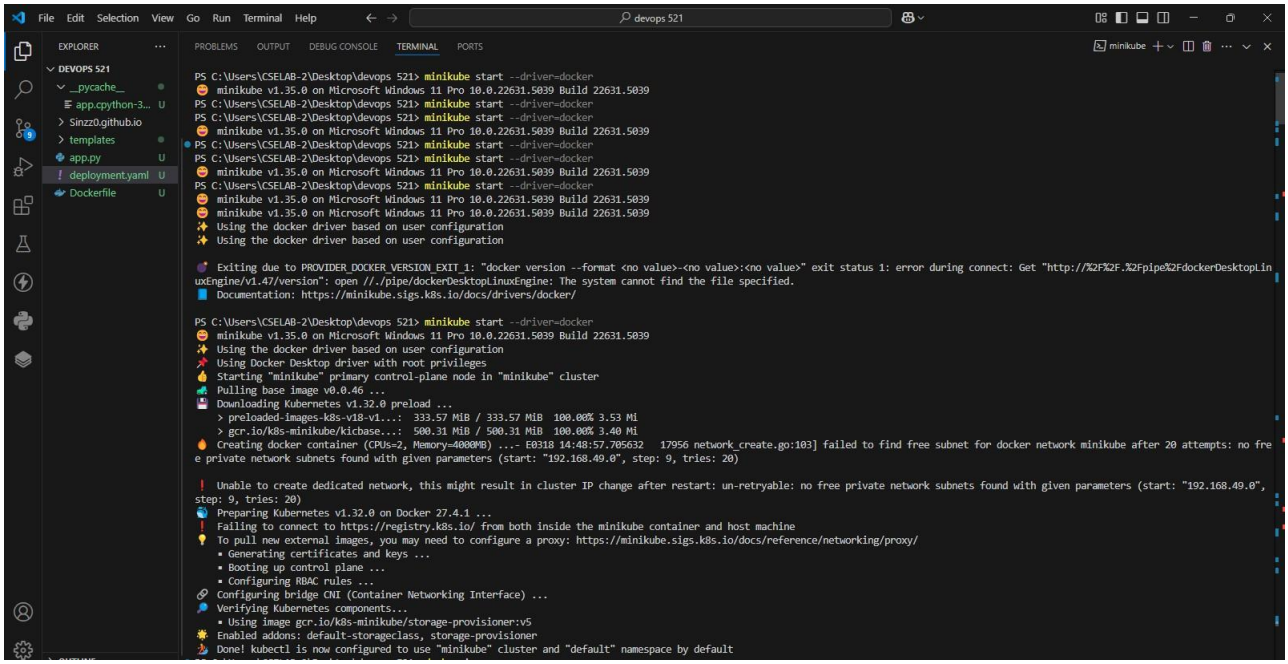
Step 1: Install and Set Up Kubernetes on Windows

1. **Enable Kubernetes in Docker Desktop**
   - Open **Docker Desktop** → Go to **Settings** → Enable **Kubernetes** → Click **Apply & Restart**.
   - Verify Kubernetes is running using:
   - kubectl version --client

kubectl get nodes

2. **Alternatively, Install Minikube (If Not Using Docker Desktop Kubernetes)**
   - Install **Minikube**
   - Start Minikube:

minikube start --driver=docker



Step 2: Build and Push Docker Image

1. **Navigate to the application directory in Command Prompt (cmd) or PowerShell**:

cd C:\Users\YourUsername\Documents\YourApp

2. **Build the Docker image**:
3. **Tag the image for Docker Hub**:
4. **Login to Docker Hub**:
5. **Push the image to Docker Hub**:

docker push <your-dockerhub-username>/myapp:v1

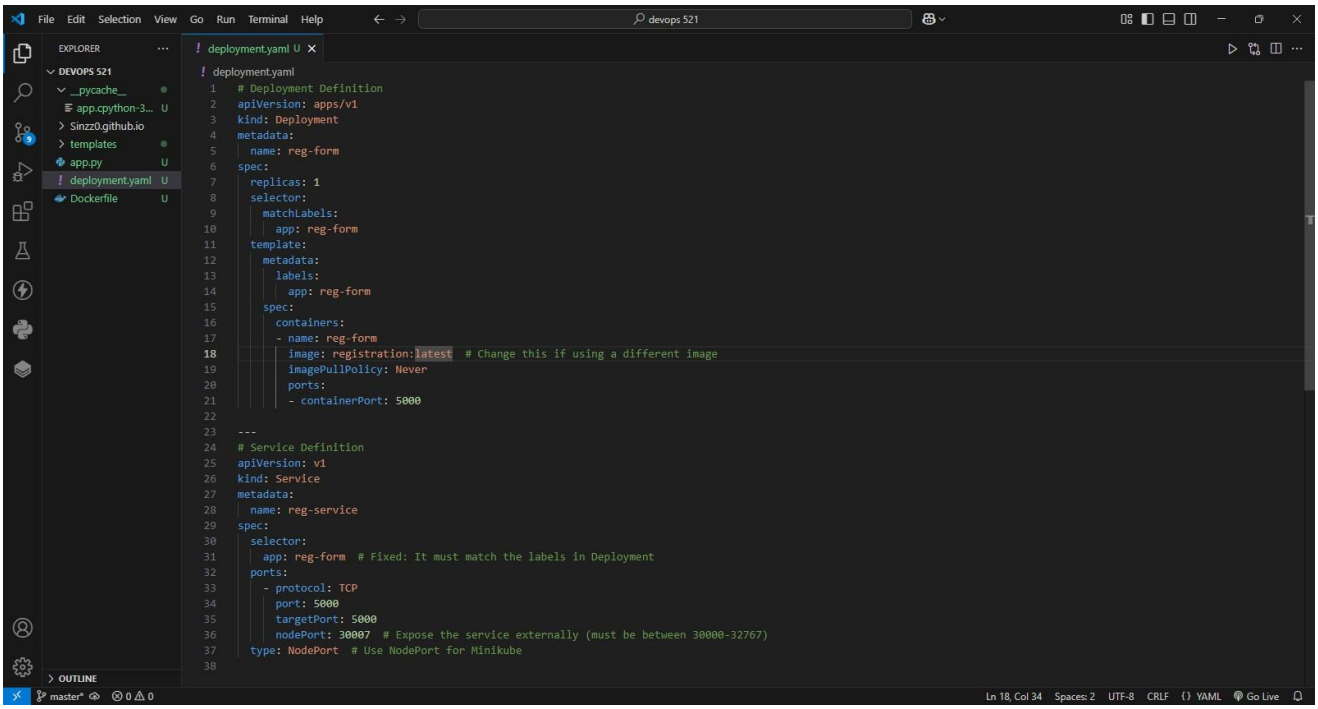
Step 3: Create Kubernetes Deployment

1. **Create a new YAML file deployment.yaml which should have both deployment and service.**
2. **Apply the deployment**:

kubectl apply -f deployment.yaml

3. **Check deployment status**:

kubectl get deployments



Step 4: Access the Application

- If using **Docker Desktop Kubernetes**, access via:

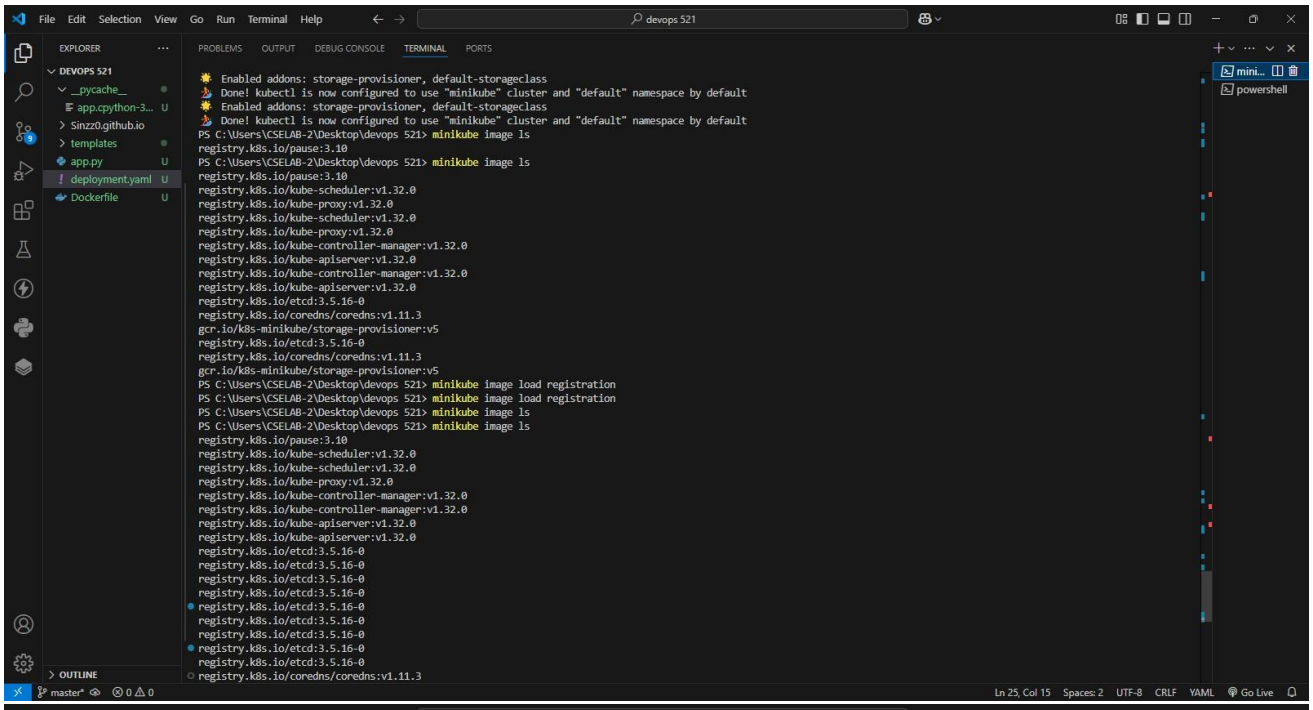
http://localhost

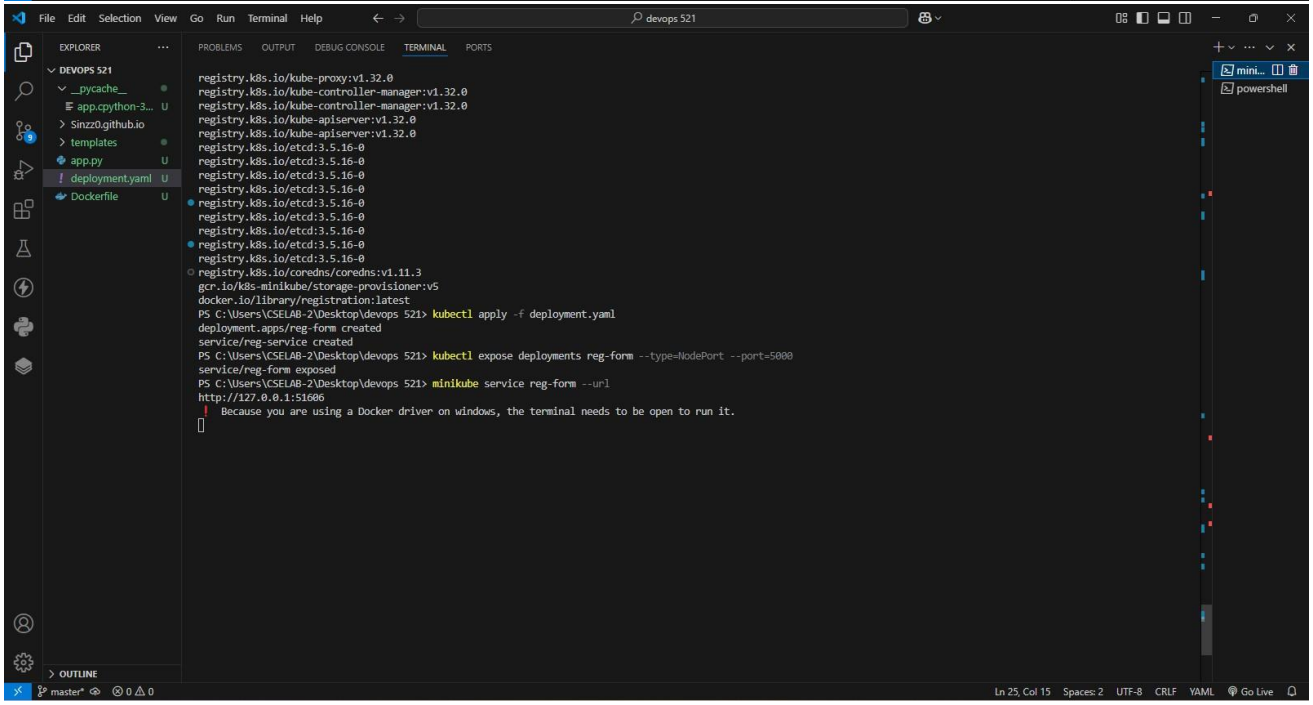
- If using **NodePort**, get the port and access via:

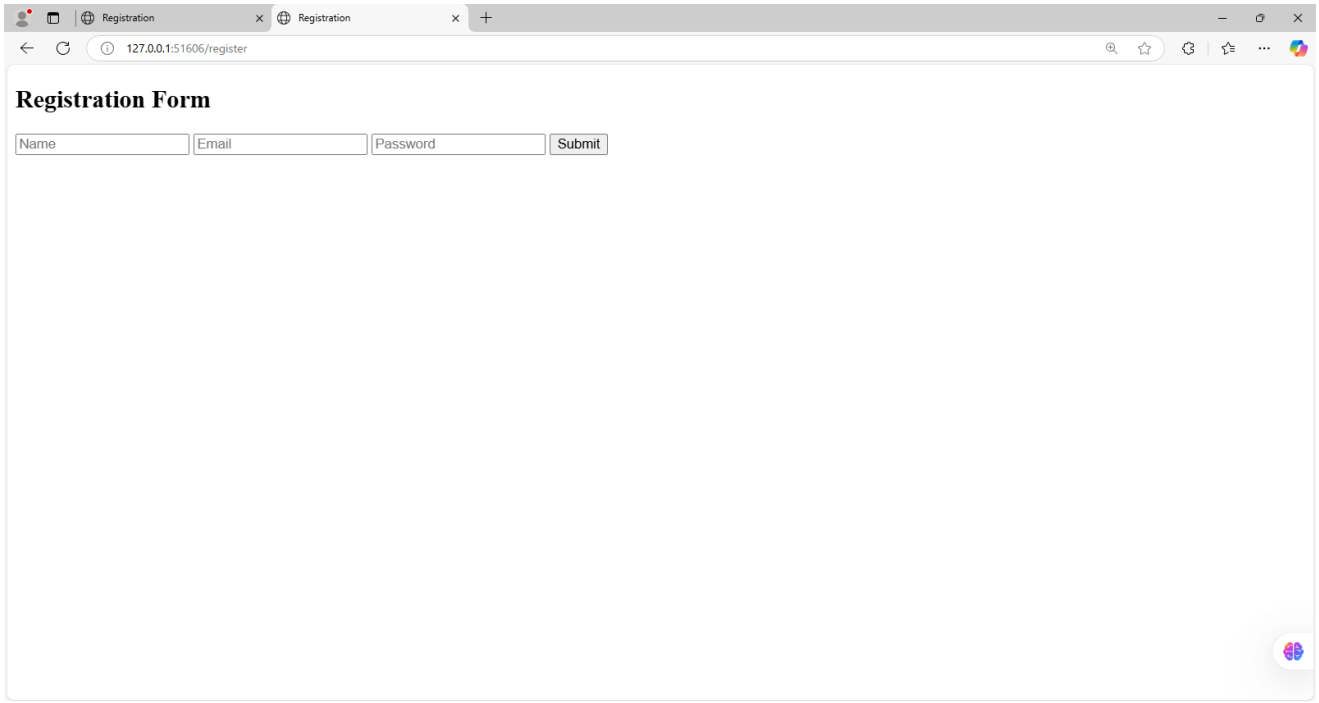
kubectl get svc

- Then open in the browser:

http://localhost:<PORT>







Hence, we successfully Automated the process of running a containerized application developed in **Exercise 7** using **Kubernetes**.

EXPERIMENT NO: 10 **DATE:**

Aim:
Install and explore **Selenium** for automated testing.

Description:
Selenium is an open-source framework for **automating web applications** across different browsers. This experiment focuses on **installing Selenium**, setting up WebDriver, and running a basic test script.

Step 1: Install Python & Selenium

1. Check if Python is Installed

Run the following command in **Command Prompt (cmd)**:

python --version

✅ If Python is not installed, download it from:

🔗 Python Official Website

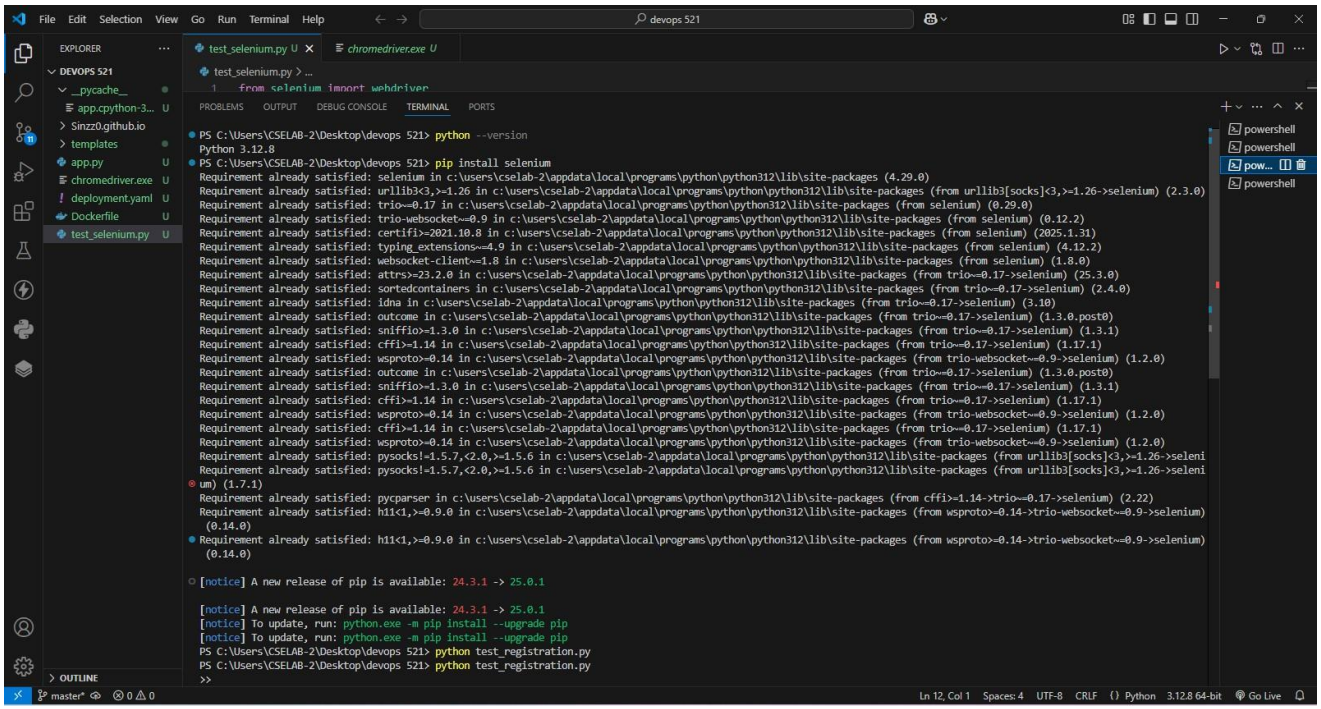
2. Install Selenium Using pip

pip install selenium

3. Verify Selenium Installation

python -c "import selenium; print(selenium.__version__)"

✅ If no error occurs, Selenium is installed successfully.



Step 2: Install WebDriver (ChromeDriver or GeckoDriver)

To control a web browser, Selenium requires a WebDriver.

- Download ChromeDriver from:
  🔗 ChromeDriver Official Site
- Place the chromedriver.exe file in the same folder as your script or **add its path to the system environment variables**.

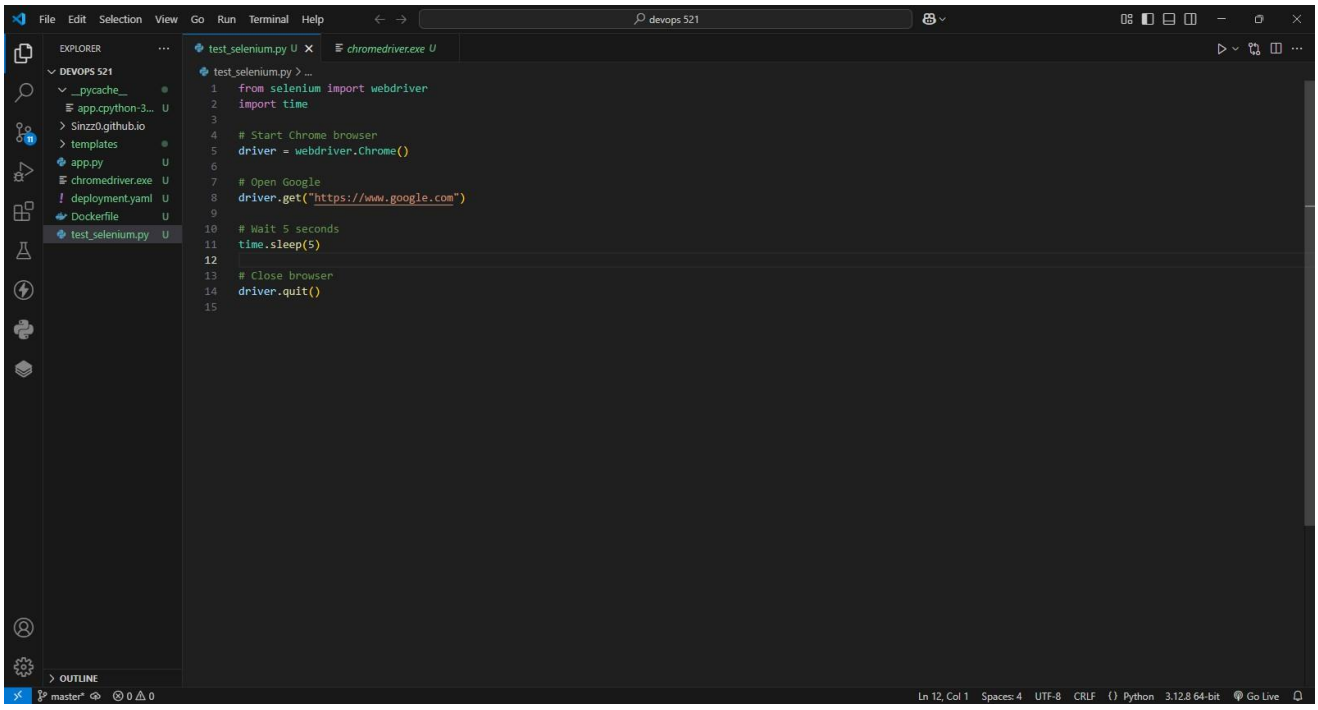
Stable

Version: 134.0.6998.88 (r1415337)

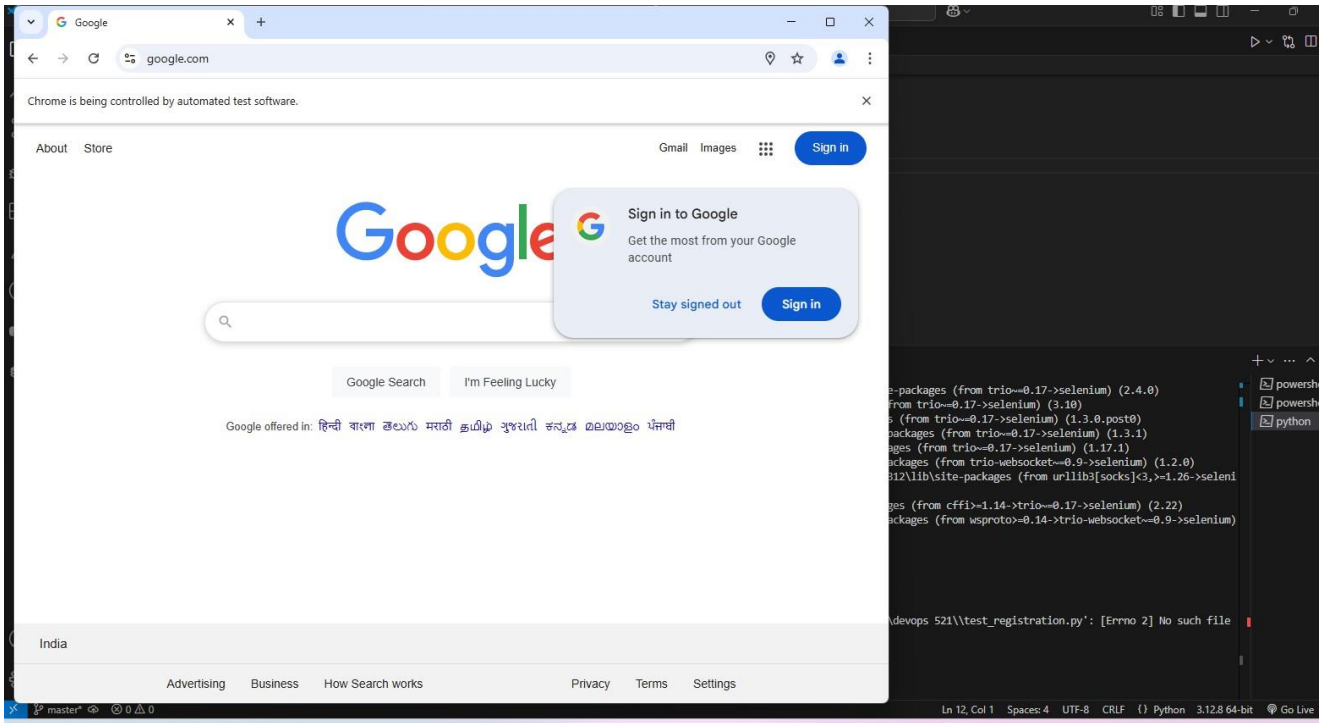
Binary	Platform	URL	HTTP status
chrome	linux64	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/linux64/chrome-linux64.zip	200
chrome	mac-arm64	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/mac-arm64/chrome-mac-arm64.zip	200
chrome	mac-x64	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/mac-x64/chrome-mac-x64.zip	200
chrome	win32	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/win32/chrome-win32.zip	200
chrome	win64	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/win64/chrome-win64.zip	200
chromedriver	linux64	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/linux64/chromedriver-linux64.zip	200
chromedriver	mac-arm64	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/mac-arm64/chromedriver-mac-arm64.zip	200
chromedriver	mac-x64	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/mac-x64/chromedriver-mac-x64.zip	200
chromedriver	win32	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/win32/chromedriver-win32.zip	200
chromedriver	win64	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/win64/chromedriver-win64.zip	200
chrome-headless-shell	linux64	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/linux64/chrome-headless-shell-linux64.zip	200
chrome-headless-shell	mac-arm64	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/mac-arm64/chrome-headless-shell-mac-arm64.zip	200
chrome-headless-shell	mac-x64	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/mac-x64/chrome-headless-shell-mac-x64.zip	200
chrome-headless-shell	win32	https://storage.googleapis.com/chrome-for-testing-public/134.0.6998.88/win32/chrome-headless-shell-win32.zip	200

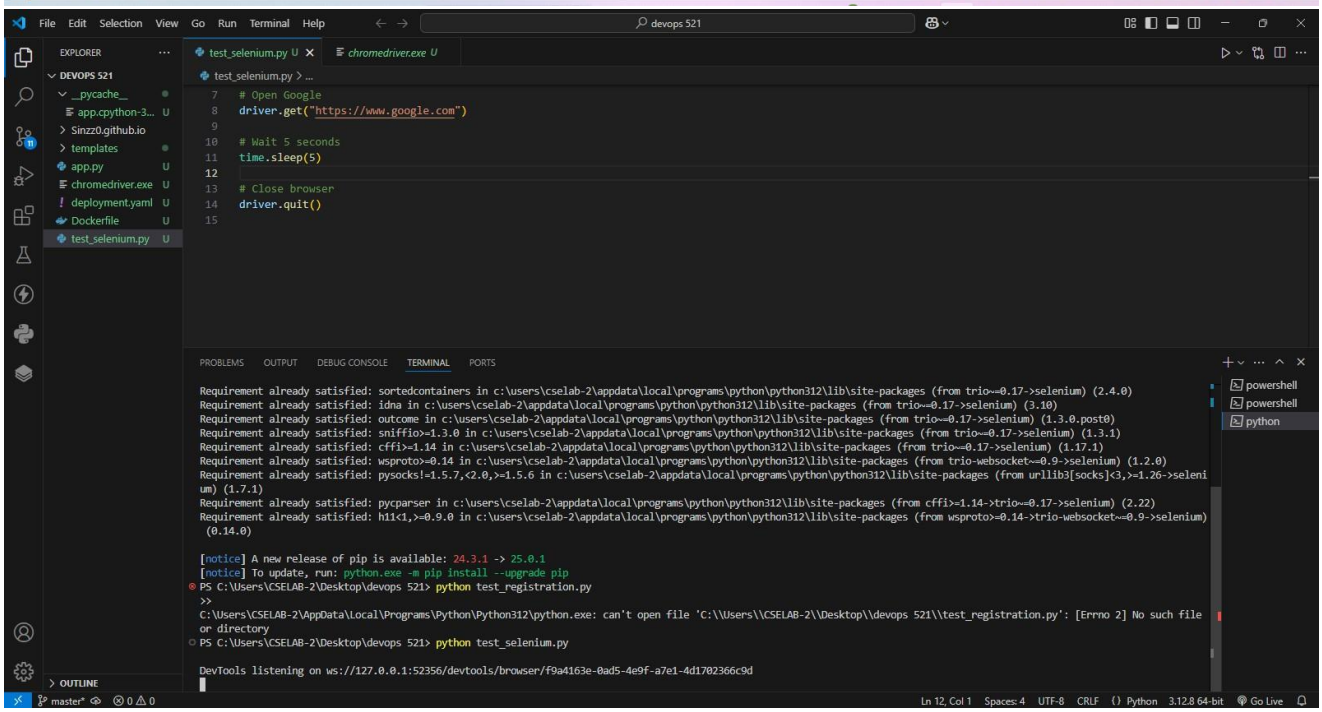
Step 3: Test Selenium Installation

Run the following Python script to check if Selenium can open a browser:



✅ **If Chrome opens and loads Google, Selenium is working!**





Chrome successfully opened and closed after 5 seconds.

Conclusion:

This experiment demonstrates the **installation and setup of Selenium**, enabling automated web browser interactions.

EXPERIMENT NO: 11 **DATE:**

Aim:

Write a JavaScript test using **Selenium** in **VS Code**.

Description:

Selenium is a powerful framework for **automating web applications**. This experiment involves setting up **Node.js**, installing **Selenium WebDriver**, writing a JavaScript Selenium test, and executing it in **VS Code**.

Step 1: Install Node.js & Selenium WebDriver

1. Check if Node.js is Installed

Run the following command in **Command Prompt (cmd)**:

node -v

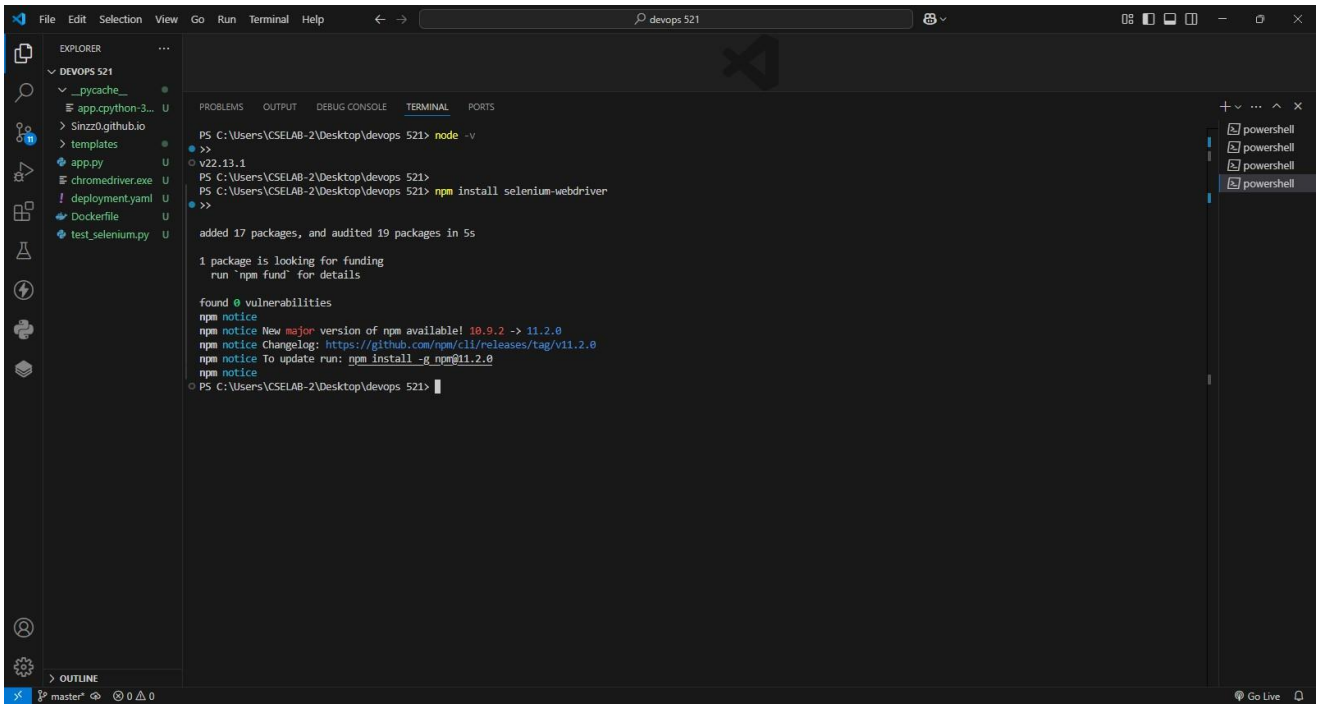
✅ If Node.js is not installed, download it from:

🔗 Node.js Official Site

2. Install Selenium WebDriver

In the **VS Code terminal**, run:

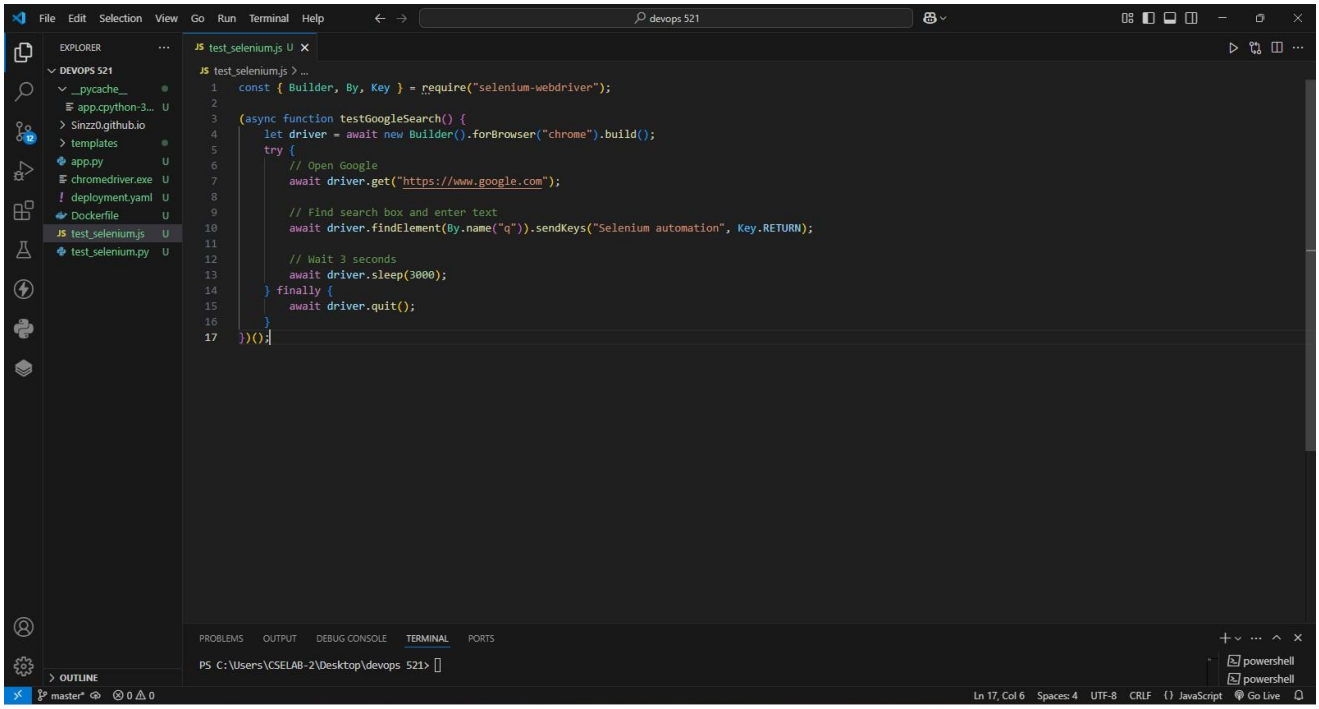
npm install selenium-webdriver



Step 2: Write a Basic JavaScript Selenium Test

Create a JavaScript file:

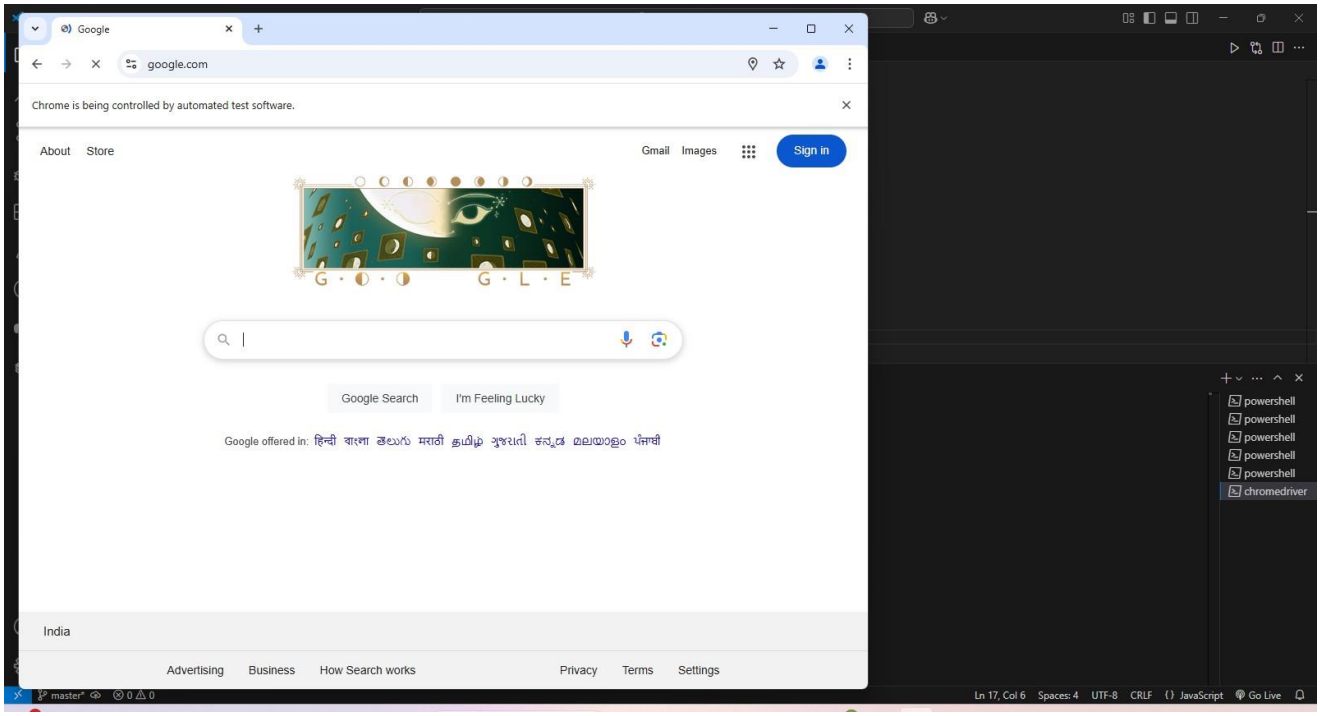
Create a new file named test_selenium.js in VS Code.

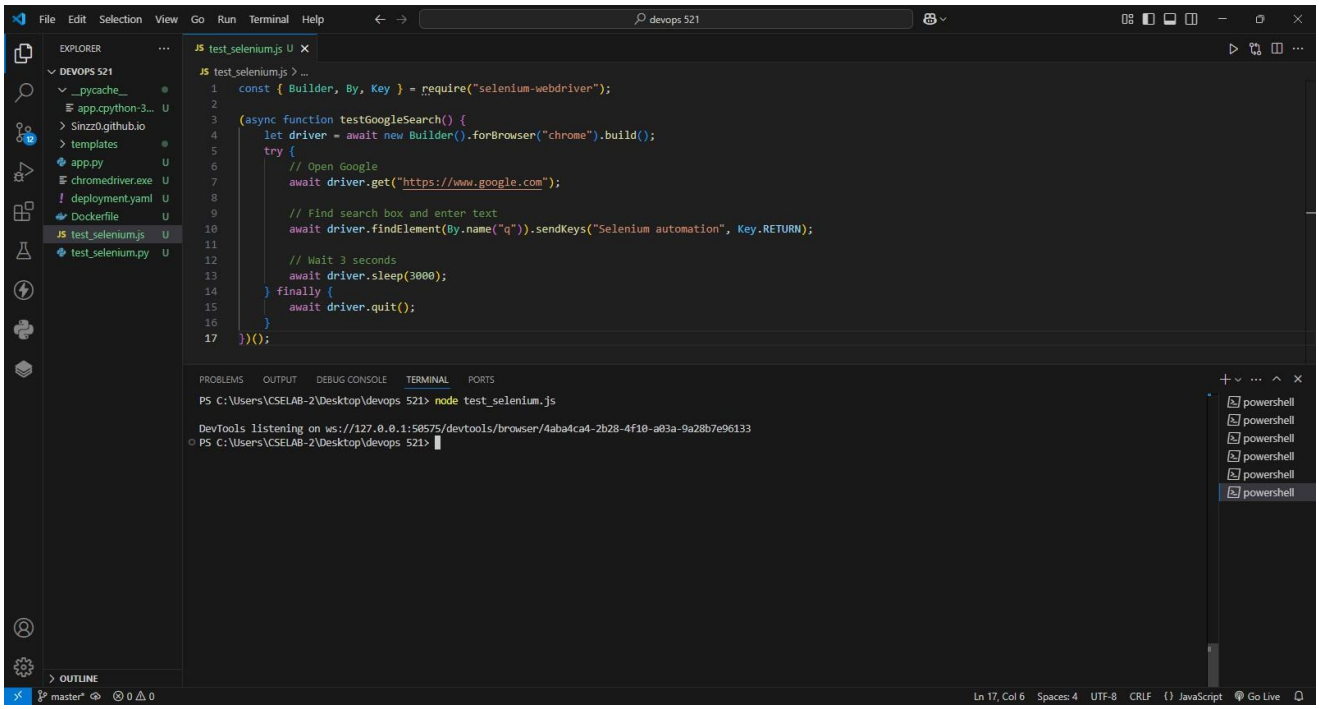


Step 3: Run the JavaScript Selenium Test

Execute the test by running the following command in **VS Code terminal**:

node test_selenium.js





☑ Behavior:

- Chrome opens.
- Google loads
- The browser closes automatically.

Conclusion:

This experiment demonstrates **writing and executing a Selenium automation script in JavaScript using VS Code**.