DEVOPS LAB EXTERNAL

SET – 1

1. Create an HTML registration form (username, password, confirm password) and upload it to GitHub.

```
<html>
<head>
        <title> EVENT REGISTRATION </title>
</head>

<body>
<form id="regform">
        <h2> Event Registration form</h2>
        <label>Email : </label>
        <input type="email" id="em" placeholder="dee@gmail.com"><br>
        <label>Password : </label>
        <input type="password" id="pw" placeholder="12345678"><br>
        <label>Confirm pw : </label>
        <input type="password" id="cpw" placeholder="12345678"><br><br>
        <button onclick="register()">Register</button>
</form>

<script>
function register(){
let email=document.getElementById("em").value;
let password=document.getElementById("pw").value;
let cpassword=document.getElementById("cpw").value;

if(email=="" || password=="" || cpassword==""){
        alert("Fill all fields!");
        return false;
}
if(!email.includes("@")|| !email.includes(".")){
        alert("Invalid email");
        return false;
}
if(password.length<8){
        alert("Password too short");
        return false;
}
if(password!=cpassword){
        alert("Passwords don't match");
        return false;
}
alert("Registration successful");
}
</script>

<style>
        body{
                text-align:center;
                background-color:#f6f6f6;
        }
</style>

</body>

</html>
```

Git hub upload

Start menu → type **Credential Manager** → Open → **Windows Credentials**

Delete ONLY these two entries:

- `git:https://deepthikadaveru@github.com`
- `git:https://github.com`

(Click Remove for both.)

```
cd /d/4ext
// rm -rf .git

git config --global user.name "deepthikada"
git config --global user.email "kdeepthi_cse2305f6@mgit.ac.in"

git init
git add .
git commit -m "first commit"
git remote add origin https://github.com/deepthikada/event-registration.git
git branch -M main
git push -u origin main
```

LOGIN

2. Create a Docker Compose setup to run two simple applications together.

```
mkdir compose-demo
cd compose-demo
```

```
compose-demo/
│   app1.py
│   app2.py
│   docker-compose.yml
```

app1.py

```
print("This is App 1 running inside a container")
```

app2.py

```
print("This is App 2 running inside a container")
```

docker-compose.yml

```
version: "3"

services:
  app1:
    image: python:3.9
    command: ["python", "app1.py"]
    volumes:
      - ./app1.py:/app1.py

  app2:
    image: python:3.9
    command: ["python", "app2.py"]
    volumes:
      - ./app2.py:/app2.py
```

CMD TO EXEC

```
docker-compose up
```

```
app1  | This is App 1 running inside a container
app2  | This is App 2 running inside a container
```

```
docker-compose down
```

This stops and removes all containers created by the compose file.

3. Create a Jenkins Freestyle job that prints “Hello STUDENT_NAME, Welcome to Jenkins!” using a string parameter.

CMD as admin
net stop jenkins
Start → Notepad → Right-click → *Run as Administrator*
Open C:\ProgramData\Jenkins\.jenkins\config.xml

<useSecurity>true</useSecurity>

To false

net start jenkins

http://localhost:8080

New project -> Freestyle ->
This project is parameterized
Add param-> String parameter

STUDENT_NAME
Deepthi
Desc : Name

Add build step
Execute Windows Batch command
echo Hello %STUDENT_NAME%, Welcome to Jenkins!

Save
Build with parameters
Build

View console output

SET – 2

1. Create an HTML feedback/contact form and push it to GitHub.

```
<html>
<head>
        <title> FEEDBACK FORM </title>
</head>

<body>
<form id="feedform">
<h2> Feedback / Contact Form </h2>

<label>Name : </label>
<input type="text" id="nm" placeholder="Deepthi"><br>

<label>Email : </label>
<input type="email" id="em" placeholder="dee@gmail.com"><br>

<label>Message : </label>
<textarea id="msg" placeholder="Write your feedback..." rows="4"></textarea><br><br>

<button onclick="sendFeedback()">Submit</button>
</form>

<script>
function sendFeedback(){
   let name = document.getElementById("nm").value;
   let email = document.getElementById("em").value;
   let message = document.getElementById("msg").value;

   if(name=="" || email=="" || message==""){
      alert("Fill all fields!");
      return false;
   }
   if(!email.includes("@") || !email.includes(".")){
      alert("Invalid email");
      return false;
   }
   if(message.length < 5){
      alert("Message too short");
      return false;
   }

   alert("Feedback submitted successfully");
}
```

```
</script>

<style>
	body{
		text-align:center;
		background-color:#f6f6f6;
	}
</style>

</body>

</html>
```

```
Cd set2q1
Git init
Git add .
Git commit -m “c1”
Git remote add origin http://github.com/deepthikada/set2.git
Git branch -M main
Git push -u origin main
```

2. Develop a containerized web application in Docker consisting of Registration + Login form.

Index.html

```
<html>
<head>
        <title>Registration</title>
</head>

<body>
<form id="regform">
<h2>Event Registration</h2>

<label>Email : </label>
<input type="email" id="em" placeholder="dee@gmail.com"><br>

<label>Password : </label>
<input type="password" id="pw" placeholder="12345678"><br>

<label>Confirm pw : </label>
<input type="password" id="cpw" placeholder="12345678"><br><br>

<button onclick="register()">Register</button><br><br>

<a href="login.html">Already have an account? Login</a>
</form>

<script>
function register(){
   event.preventDefault();
   let email=document.getElementById("em").value;
   let password=document.getElementById("pw").value;
   let cpassword=document.getElementById("cpw").value;

   if(email=="" || password=="" || cpassword==""){
      alert("Fill all fields!");
      return false;
   }
   if(!email.includes("@")|| !email.includes(".")){
      alert("Invalid email");
      return false;
   }
   if(password.length<8){
      alert("Password too short");
      return false;
```

```
    }
    if(password!=cpassword){
        alert("Passwords don't match");
        return false;
    }
    localStorage.setItem("email", email);
    localStorage.setItem("password", password);
    alert("Registration successful");
}
</script>

<style>
        body{
                text-align:center;
                background-color:#f6f6f6;
        }
</style>

</body>
</html>
```

Login.html

```
<html>
<head>
        <title>Login</title>
</head>

<body>
<form id="logform">
<h2>Login</h2>

<label>Email : </label>
<input type="email" id="lem" placeholder="dee@gmail.com"><br>

<label>Password : </label>
<input type="password" id="lpw" placeholder="12345678"><br><br>

<button onclick="login()">Login</button><br><br>

<a href="index.html">Create new account</a>
</form>

<script>
function login(){
    event.preventDefault();
    let email=document.getElementById("lem").value;
    let password=document.getElementById("lpw").value;

    let regEmail = localStorage.getItem("email");
    let regPw = localStorage.getItem("password");

    if(email=="" || password==""){
        alert("Fill all fields!");
        return;
    }
    if(email===regEmail && password===regPw){
        alert("Login successful");
    }
    else{
        alert("Invalid credentials");
    }
}
</script>

<style>
        body{
                text-align:center;
                background-color:#f6f6f6;
        }
</style>

</body>
</html>
```

Dockerfile

```
FROM nginx:alpine

COPY index.html /usr/share/nginx/html/index.html
COPY login.html /usr/share/nginx/html/login.html

EXPOSE 80
```

EXECUTE CMDs

```
docker build -t eventapp .
docker run -p 8081:80 eventapp
```

http://localhost:8081
http://localhost:8081/login.html

3. Create a Jenkins Pipeline with Build → Test → Deploy stages, each printing its stage name.

```
pipeline {
    agent any

    stages {

        stage('Build') {
            steps {
                echo "Build Stage"
            }
        }

        stage('Test') {
            steps {
                echo "Test Stage"
            }
        }

        stage('Deploy') {
            steps {
                echo "Deploy Stage"
            }
        }
    }
}
```

SET – 3

1. Create a Git branching story project where each branch adds a chapter and merge all into a final book.html.

Step 1: Create Project Folder

```
mkdir story-branches
cd story-branches
```

Step 2: Initialize project

```
git init
```

Step 3: Create Base Story File

```
echo "<html><body><h1>My Storybook</h1></body></html>" > book.html
git add book.html
git commit -m "Initial story template"
```

Step 4: Create Branch for chp1

```
git checkout -b chapter1
```

Edit `book.html` and add:

```
    <h2>Chapter 1: The Mysterious Forest</h2>
    <p>Once upon a time, deep in the forest...</p>
```

Save → then:

```
    git add book.html
    git commit -m "Add Chapter 1"
```

Step 5: Create Branch for chp2

```
git checkout main
git checkout -b chapter2
```

Edit `book.html` and add:

```
<h2>Chapter 2: The Hidden Cave</h2>
<p>The hero found a cave glowing with strange light...</p>
```

Save → then:

```
git add book.html
git commit -m "Add Chapter 2"
git checkout main
```

Step 6: Merge Branches Into Main

Merge chptrs:

```
git merge chapter1
git merge chapter2
```

If Merge Conflict Occurs

Fix file manually → remove markers → keep both chapters.

Then:

```
git add book.html
git commit -m "Resolved merge conflict"
```

Step 7: Create GitHub Repository

Go to:
https://github.com/new

Name:

```
story-branches
```

Step 8: Connect Local Folder to GitHub

```
git branch -M main
git remote add origin
https://github.com/deepthikada/story-branches.git
```

Step 9: Push Entire EXP-3 Project

```
git push -u origin main

git push origin chapter1
git push origin chapter2
```

2. Develop a simple Java user-interactive containerized application (e.g., calculator) using Docker.

Calculator.java

```
import java.util.Scanner;

public class Calculator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Simple Calculator");

        System.out.print("Enter first number: ");
        int a = sc.nextInt();

        System.out.print("Enter second number: ");
        int b = sc.nextInt();

        System.out.print("Choose (+  -  *  /): ");
        char op = sc.next().charAt(0);

        int result = 0;

        if(op == '+') result = a + b;
        else if(op == '-') result = a - b;
        else if(op == '*') result = a * b;
        else if(op == '/' && b != 0) result = a / b;
        else System.out.println("Invalid operation");
        System.out.println("Result: " + result);
    }
}
```

Dockerfile

```
FROM eclipse-temurin:17

WORKDIR /app

COPY Calculator.java /app

RUN javac Calculator.java

CMD ["java", "Calculator"]
```

```
docker build -t java-calc .

docker run -it java-calc
```

3. Automate deployment of that containerized application using Kubernetes Deployment + Service.

```
public class Calculator {
    public static void main(String[] args) {
        System.out.println("Simple Calculator Running in 
Kubernetes...");

        int a = 10;
        int b = 20;

        System.out.println("Sample Addition: " + a + " + " + b + " = " 
+ (a + b));
        System.out.println("Container deployed successfully!");
    }
}
```

```
docker build -t set3q2 .

docker tag set3q2 kdeepthi6002/set3q2

docker push kdeepthi6002/set3q2
```

Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: calc-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: calc
  template:
    metadata:
      labels:
        app: calc
    spec:
      containers:
      - name: calc-container
        image: kdeepthi6002/set3q2
        imagePullPolicy: Always
```

```
apiVersion: v1
kind: Service
metadata:
  name: calc-service
spec:
  type: NodePort
  selector:
    app: calc
  ports:
  - port: 8080
    targetPort: 8080
    nodePort: 30081
```

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml

kubectl get pods

kubectl logs -l app=calc

kubectl get deployments

kubectl get svc
```

SET – 4

1. Build an HTML page with a profile card (photo, name, description) and push it to GitHub.

```
<html>
<head>
   <title>Profile Card</title>
</head>

<body>
   <div id="card">
      <h2>My Profile</h2>

      <img
src="https://tse2.mm.bing.net/th/id/OIP.7cRYFyLoDEDh4sRtM73vvwHaDg?pid=Api&P=0&h
=180" width="120" height="120" ;"><br><br>

      <label>Name : </label>
      <p>Deepthi Kadaveru</p>

      <label>Description : </label>
      <p>5th semester CSE student.</p>
   </div>
```

```
<style>
    body{
        text-align:center;
        background-color:#f6f6f6;
    }
</style>

</body>
</html>
```

```
git init
git add .
git commit -m "Profile card added"
git remote add origin https://github.com/<your-username>/<your-repo>.git
git branch -M main
git push -u origin main
```

2. List the installation steps for Docker and demonstrate container/content management commands.

Docker Installation Steps (Simple Points)

1. Download **Docker Desktop** from the official website.
2. Run the installer and follow the on-screen steps.
3. On Windows, enable **“Use WSL 2 backend”** when prompted.
4. Complete the installation and restart your system.
5. Open Docker Desktop and wait until it shows **“Docker Engine Running.”**

Verify installation using:

```
docker --version
```

Docker Commands (Tables Grouped by Category)

---

1. Basic Docker Commands

Command	Description	Example
`docker --version`	Check Docker version	`docker --version`
`docker info`	System-wide Docker details	`docker info`
`docker help`	List all Docker commands	`docker help`

---

2. Image Management Commands

Command	Description	Example
`docker images`	List images stored locally	`docker images`
`docker pull <image>`	Download image from Docker Hub	`docker pull nginx`
`docker rmi <image>`	Remove an image	`docker rmi nginx`

---

3. Container Management Commands

Command	Description	Example
`docker ps`	List running containers	`docker ps`
`docker ps -a`	List all containers	`docker ps -a`
`docker run <image>`	Create + start a container	`docker run nginx`
`docker start <id>`	Start a stopped container	`docker start 6a4c2b`

`docker stop <id>`	Stop a running container	`docker stop 6a4c2b`
`docker rm <id>`	Remove a container	`docker rm 6a4c2b`

4. Container/Content Interaction Commands

Command	Description	Example
`docker exec -it <id> bash`	Enter inside container shell	`docker exec -it 6a4c2b bash`
`docker logs <id>`	View container logs	`docker logs 6a4c2b`
`docker inspect <id>`	Detailed info about container	`docker inspect 6a4c2b`
`docker stats`	Live resource usage	`docker stats`

5. Docker Network Commands

Command	Description	Example
`docker network ls`	List all networks	`docker network ls`
`docker network create <name>`	Create a new network	`docker network create mynet`
`docker network inspect <name>`	View network details	`docker network inspect mynet`

6. Docker Volume Commands

Command	Description	Example
`docker volume ls`	List volumes	`docker volume ls`
`docker volume create <name>`	Create a new volume	`docker volume create data1`
`docker volume inspect <name>`	View volume details	`docker volume inspect data1`

`docker volume rm <name>`	Delete a volume	`docker volume rm data1`

---

7. Docker Compose Commands (if installed)

Command	Description	Example
`docker-compose up`	Start services	`docker-compose up`
`docker-compose down`	Stop and remove services	`docker-compose down`
`docker-compose ps`	List compose-managed containers	`docker-compose ps`

3. Run a Java Selenium script to test Google and MGIT website.

Add selenium jar files to libraries

Replace chrome driver path

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class Set4q3 {
   public static void main(String[] args) {

      // Path to chromedriver.exe (if needed)
      String setProperty = System.setProperty("webdriver.chrome.driver", "C:\\Users\\K
RAJU\\Downloads\\chromedriver-win64\\chromedriver-win64\\chromedriver.exe");

      WebDriver driver = new ChromeDriver();

      try {
         // Test Google
         driver.get("https://www.google.com");
         System.out.println("Google Title: " + driver.getTitle());
         Thread.sleep(2000);

         // Test MGIT
```

```
            driver.get("https://mgit.ac.in/");
            System.out.println("MGIT Title: " + driver.getTitle());
            Thread.sleep(2000);

        } catch (InterruptedException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            driver.quit();
        }
    }
}
```

Selenium java client library https://www.selenium.dev/downloads/

chrome driver https://googlechromelabs.github.io/chrome-for-testing/

SET – 5

1. Develop an HTML page showing your favorite books/movies and push it to GitHub.

```
<html>
<head>
    <title>My Favorites</title>
</head>

<body>
<h2>My Favorite Books & Movies</h2>

<h3>Books</h3>
<p>The Palace of Illusions</p>
<p>A Good Girl's Guide to Murder</p>
<p>The Boyfriend</p>

<h3>Movies</h3>
<ul>
<li>m1</li>
<li>m2</li>
<li>m3</li>
</ul>

</body>
```

```
</html>
```

```
git init
git add .
git commit -m "c1"
git remote add origin https://github.com/deepthikada/set5.git
git branch -M main
git push -u origin main
```

2. Create a branch bio-update, add 3 hobbies to a file and push the branch.

Hobbies.txt

```
Eat
Sleep
Scroll
```

```
git add hobbies.txt
git commit -m "c2"
git push -u origin bio-update
git checkout main
git merge bio-update
git push origin main
```

3. Create and run a Simple Python containerized application (e.g., number guessing) using Docker.

guess.py

```
import random

print("Number Guessing Game")
num = random.randint(1, 10)

guess = int(input("Guess a number (1-10): "))

if guess == num:
    print("Correct!")
else:
    print("Wrong, the number was:", num)
```

Dockerfile

```
FROM python:3.10
WORKDIR /app
COPY guess.py .
CMD ["python", "guess.py"]
docker build -t py-guess .
docker run -it py-guess
```

SET – 6

1. Design an HTML Weekly Timetable and upload it to GitHub.

```
<html>
<head>
  <title>Weekly Timetable</title>
</head>

<body>
<h2>Weekly Timetable</h2>

<table border="1" cellpadding="6" cellspacing="0" style="margin:auto;">
  <tr>
    <th>Day</th>
    <th>9–10</th>
    <th>10–11</th>
    <th>11–12</th>
  </tr>
  <tr>
    <td>Mon</td>
    <td>Math</td>
    <td>DSA</td>
    <td>Break</td>
  </tr>
  <tr>
    <td>Tue</td>
    <td>Java</td>
    <td>DBMS</td>
    <td>Break</td>
    </tr>
    <tr>
        <td>Wed</td>
        <td>COA</td>
        <td>OS</td>
        <td>Break</td>
    </tr>
</table>

<style>
    body{
        text-align:center;
        background-color:#f6f6f6;
    }
</style>

</body>
</html>
```

```
git init
git add timetable.html
git commit -m "Add weekly timetable"
git remote add origin https://github.com/<your-username>/<your-repo>.git
git branch -M main
git push -u origin main
```

2. Write steps to integrate Kubernetes with Docker, and explore basic kubectl commands.

1. **Install Docker Desktop**
   (Because it includes Kubernetes support.)

2. **Open Docker Desktop Settings**

3. **Go to “Kubernetes” section**
   Yes, it actually sits there waiting for attention.

4. **Enable “Enable Kubernetes” checkbox**

5. **Click “Apply & Restart”**

6. Wait for it to finish installing:
   Docker will download required Kubernetes components and start the cluster.

Verify installation using:

```
kubectl version --client
kubectl cluster-info
```

7.
8. Kubernetes is now running as a **single-node cluster** inside Docker.

And that's literally all the "integration" they want.
Do this, write it in your lab record, collect your marks.

Basic kubectl Commands (simple table)

Cluster Information

Command	Description
`kubectl version`	Check Kubernetes version
`kubectl cluster-info`	Shows cluster details
`kubectl get nodes`	List cluster nodes
`kubectl get pods -A`	List all pods in all namespaces

Working With Pods

Command	Description	Example
`kubectl run <name> --image=<img>`	Create a pod	`kubectl run mypod --image=nginx`
`kubectl get pods`	List pods	`kubectl get pods`

`kubectl describe pod <name>`	Details of a pod	`kubectl describe pod mypod`
`kubectl delete pod <name>`	Delete a pod	`kubectl delete pod mypod`

Working With Deployments

Command	Description	Example
`kubectl create deployment <name> --image=<img>`	Create deployment	`kubectl create deployment web --image=nginx`
`kubectl get deployments`	List deployments	`kubectl get deployments`
`kubectl scale deployment <name> --replicas=<n>`	Scale replicas	`kubectl scale deployment web --replicas=3`
`kubectl delete deployment <name>`	Remove deployment	`kubectl delete deployment web`

Services (Exposing Pods)

Command	Description	Example
`kubectl expose deployment <name> --type=NodePort --port=<p>`	Expose deployment/service	`kubectl expose deployment web --type=NodePort --port=80`
`kubectl get svc`	List services	`kubectl get svc`

Configuration & Debugging

Command	Description
`kubectl logs <pod>`	View pod logs
`kubectl exec -it <pod> -- bash`	Enter pod shell

`kubectl apply -f file.yaml`	Apply config file
`kubectl delete -f file.yaml`	Delete resources from file

3. Create a job that runs every 2 minutes and prints the system date/time in Jenkins.

```
pipeline {
   agent any

   triggers {
      cron('H/2 * * * *')
   }

   stages {
      stage('Print Date and Time') {
         steps {
            script {
               println "Current Date & Time: " + new Date()
            }
         }
      }
   }
}
```

Enable **Build periodically**

Enter schedule:

```
H/2 * * * *
```

SET – 7

1. Create an HTML Resume page (Education, Skills, Projects) and push it.

```
<html>
<head>
   <title>My Resume</title>
</head>

<body>
<h2>My Resume</h2>

<h3>Education</h3>
<p>B.Tech CSE (4th Semester)</p>

<h3>Skills</h3>
<p>Java, Python, HTML, CSS, MySQL</p>

<h3>Projects</h3>
<p>StationCompass, Online Test App, College Predictor</p>

<style>
   body{
      text-align:center;
      background-color:#f6f6f6;
   }
</style>

</body>
</html>
```

```
git init
git add resume.html
git commit -m "Added resume page"
git remote add origin https://github.com/<your-username>/<your-repo>.git
git branch -M main
git push -u origin main
```

2. Build a containerized version of the application from using a Dockerfile.

```
FROM nginx:alpine

COPY resume.html /usr/share/nginx/html/index.html

EXPOSE 8082

docker build -t set7 .
```

3. Deploy the image using Kubernetes

```
docker tag set7 kdeepthi6002/set7

docker push kdeepthi6002/set7

docker login
```

Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: resume-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: resume
  template:
    metadata:
      labels:
        app: resume
    spec:
      containers:
      - name: resume-container
        image: kdeepthi6002/set7
        imagePullPolicy: Always
        ports:
        - containerPort: 80
```

Service.yaml
```
apiVersion: v1
kind: Service
metadata:
  name: resume-service
spec:
  type: NodePort
  selector:
    app: resume
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30082
```

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml

kubectl get pods

kubectl get svc
```

SET – 8

1. Create a simple HTML page to display tables and lists (any structure) and push it to GitHub.

```
<html>
<head>
    <title>Tables & Lists</title>
</head>

<body>
<h2>Simple Table</h2>

<table border="1" cellpadding="6" cellspacing="0" style="margin:auto;">
    <tr><th>Name</th><th>Marks</th></tr>
    <tr><td>Maths</td><td>90</td></tr>
    <tr><td>Science</td><td>85</td></tr>
</table>

<h2>Simple List</h2>

<ul>
```

```
    <li>Reading</li>
    <li>Coding</li>
    <li>Music</li>
</ul>

<style>
    body{
        text-align:center;
        background-color:#f6f6f6;
    }
</style>
</body>
</html>
```

```
git init
git add tables.html
git commit -m "c1"
git remote add origin https://github.com/<your-username>/<your-repo>.git
git branch -M main
git push -u origin main
```

2. Build a multi-stage Git branching task where each branch stores a part of a secret message and merge into main.

```
mkdir secret-msg
cd secret-msg

echo "Secret Message Assembly" > README.txt

git init
git add .
git commit -m "Initial commit"

git branch -M main

git remote add origin https://github.com/kdeepthi6002/set8q2.git

git push -u origin main

git checkout -b part1
echo "HELLO" > part1.txt
```

```
git add part1.txt
git commit -m "Add part1"
git push -u origin part1

git checkout main
git checkout -b part2
echo "FROM" > part2.txt
git add part2.txt
git commit -m "Add part2"
git push -u origin part2

git checkout main
git checkout -b part3
echo "GIT" > part3.txt
git add part3.txt
git commit -m "Add part3"
git push -u origin part3

git checkout main
git pull origin main

git merge part1 --no-ff -m "Merge part1"
git merge part2 --no-ff -m "Merge part2"
git merge part3 --no-ff -m "Merge part3"

git push origin main

cat part1.txt part2.txt part3.txt > final_secret.txt
git add final_secret.txt
git commit -m "Add final secret message"
git push origin main
```

3. Write a JavaScript program for Calculator and test it using Selenium WebDriver.

Calculator.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Simple Calculator</title>
</head>
<body>

<h2>Simple Calculator</h2>

<input id="n1" type="number" placeholder="Num 1"><br><br>
```

```
<input id="n2" type="number" placeholder="Num 2"><br><br>

<button onclick="add()">Add</button>

<p id="result"></p>

<script>
function add() {
   let a = parseInt(document.getElementById("n1").value);
   let b = parseInt(document.getElementById("n2").value);
   document.getElementById("result").innerHTML = "Result: " + (a + b);
}
</script>

</body>
</html>
```

test.js

```
const { Builder, By } = require("selenium-webdriver");

async function testCalculator() {

   // Start Chrome
   let driver = await new Builder().forBrowser("chrome").build();

   try {
      // Open calculator
      await driver.get("file:///D:/devext/calculator.html");   // <-- Use your actual path

      // Enter numbers
      await driver.findElement(By.id("n1")).sendKeys("5");
      await driver.findElement(By.id("n2")).sendKeys("7");

      // Click Add
      await driver.findElement(By.tagName("button")).click();

      // Read result
      let result = await driver.findElement(By.id("result")).getText();
      console.log("Calculator Output:", result);

   } catch (err) {
      console.log("Error:", err);
   } finally {
      await driver.quit();
   }
}
testCalculator();
```

```
npm init -y
npm install selenium-webdriver

node test.js
```

SET – 10

1. Create an HTML page with a Pizza Order form and commit it to GitHub.

```
<html>
<head>
   <title>Pizza Order Form</title>
</head>

<body>

<h2>Pizza Order Form</h2>

<form>
   <label>Name:</label><br>
   <input type="text" placeholder="Your Name"><br><br>

   <label>Pizza Size:</label><br>
   <select>
      <option>Small</option>
      <option>Medium</option>
      <option>Large</option>
   </select><br><br>

   <label>Toppings:</label><br>
   <input type="checkbox"> Cheese
   <input type="checkbox"> Corn
   <input type="checkbox"> Paneer
   <input type="checkbox"> Olives<br><br>

   <label>Address:</label><br>
   <textarea rows="3" cols="25"></textarea><br><br>

   <button>Place Order</button>
</form>

<style>
```

```
  body{
      text-align:center;
      background-color:#f6f6f6;
  }
  form{
      display:inline-block;
      text-align:left;
  }
</style>

</body>
</html>

git init
git add pizza_order.html
git commit -m "Add pizza order form"
git remote add origin https://github.com/<your-username>/<your-repo>.git
git branch -M main
git push -u origin main
```

2. Write installation steps for Selenium and execute a JavaScript (Node.js) Selenium script for UI testing.

A. Installation Steps

1. Install Node.js

- Download from: https://nodejs.org
- Install using default settings
- Verify:

```
node -v

npm -v
```

2. Create a project folder

```
mkdir selenium-test

cd selenium-test
```

3. Initialize a Node.js project

```
npm init -y
```

4. Install Selenium WebDriver

```
npm install selenium-webdriver
```

5. Install browser (Chrome / Edge)

- Selenium Manager automatically downloads the correct driver
- No manual chromedriver required if Selenium ≥ 4.6

B. JavaScript Selenium Script (UI Testing Example)

Create:

`test.js`

```
const { Builder, By } = require("selenium-webdriver");

async function runTest() {

    let driver = await new
Builder().forBrowser("chrome").build();

    try {

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

        console.log("Page Title:", await driver.getTitle());

    } catch (error) {

        console.log("Error:", error);

    } finally {

        await driver.quit();

    }

}

runTest();
```

Run script

```
node test.js
```

3. Develop Selenium test cases for the containerized application you created in the previous questions.

Create Dockerfile

```
FROM nginx:alpine
COPY pizza_order.html /usr/share/nginx/html/index.html
EXPOSE 80
```

Build image

```
docker build -t pizzaapp .
```

Run container

```
docker run -p 8080:80 pizzaapp
```

Application URL:

```
http://localhost:8080
```

---

4. Install Selenium (Node.js)

Inside your project folder:

```
npm init -y
npm install selenium-webdriver
```

Chrome gets managed automatically by Selenium Manager.

---

5. Selenium Test Script (JavaScript)

File: `testpizza.js`

```
const { Builder, By } = require("selenium-webdriver");

async function testPizzaForm() {
    let driver = await new Builder().forBrowser("chrome").build();

    try {
        // Open Dockerized application
        await driver.get("http://localhost:8080");

        // Test Case 1: Verify title
        console.log("Title:", await driver.getTitle());

        // Test Case 2: Verify heading
        let heading = await
driver.findElement(By.tagName("h2")).getText();
        console.log("Heading:", heading);

        // Test Case 3: Enter name
        let nameInput = await
driver.findElement(By.css("input[type='text']"));
        await nameInput.sendKeys("Deepthi");

        // Test Case 4: Select pizza size
        let sizeSelect = await
driver.findElement(By.tagName("select"));
        await sizeSelect.sendKeys("Medium");

        // Test Case 5: Click a topping checkbox
        let checks = await
driver.findElements(By.css("input[type='checkbox']"));
        await checks[0].click();

        // Button check
        let btn = await
driver.findElement(By.tagName("button")).getText();
        console.log("Button Text:", btn);

    } catch (err) {
        console.log("Error:", err);
    } finally {
        await driver.quit();
    }
}

testPizzaForm();
```

6. Run Selenium Tests

```
node testpizza.js
```

7. Summary of Test Cases

Test Case No	Description
TC1	Verify the page loads through Docker (`localhost:8080`)
TC2	Check if heading `<h2>` exists
TC3	Enter text into the name input field
TC4	Select a pizza size from dropdown
TC5	Click a toppings checkbox
Extra	Validate button text

Total used: **5 test cases**

SET – 9

1. Build an HTML page with a gallery of 3–4 images with captions and upload it.

```
<!DOCTYPE html>
<html>
<head>
   <title>Gallery</title>
</head>
<body>

<h2>Image Gallery</h2>

<div>
   <figure>
      <img src="https://picsum.photos/300?1" width="200">
      <figcaption>Caption 1</figcaption>
   </figure>

   <figure>
      <img src="https://picsum.photos/300?2" width="200">
      <figcaption>Caption 2</figcaption>
   </figure>

   <figure>
      <img src="https://picsum.photos/300?3" width="200">
      <figcaption>Caption 3</figcaption>
   </figure>

   <figure>
      <img src="https://picsum.photos/300?4" width="200">
      <figcaption>Caption 4</figcaption>
   </figure>
</div>

</body>
</html>
```

```
Git init
Git add .
Git commit -m “c1”
git remote add origin https://github.com/deepthikada/set9q1.git
Git branch -M main
Git push origin main
```

2. Create a Docker image for the HTML gallery application and run it with port mapping.

Dockerfile
```
FROM nginx:alpine
COPY img.html /usr/share/nginx/html/index.html
```

```
Rename Dockerfile.txt Dockerfile
Docker build -t set9q1 .
Docker run -d -p 8083:80 set9q1
```

See localhost:8083

3. Automate the gallery application deployment using a Kubernetes Deployment + Service YAML.

Deployment.yaml
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gallery-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: gallery
  template:
    metadata:
      labels:
        app: gallery
    spec:
      containers:
      - name: gallery
        image: kdeepthi6002/set9q1
        ports:
        - containerPort: 80
```

Service.yaml
```
apiVersion: v1
kind: Service
metadata:
  name: gallery-service
spec:
  type: NodePort
```

```
 selector:
   app: gallery
 ports:
 - port: 80
   targetPort: 80
   nodePort: 30083
```

```
Docker tag set9q1 kdeepthi6002/set9q1
Docker push kdeepthi6002/set9q1
>kubectl apply -f deployment.yaml
>kubectl apply -f service.yaml
>kubectl get pods
http://localhost:30083/
```