Experiment: 1

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

Aim: To write code for a simple user registration form for an event.
Description: Here's an example of a simple user registration form using Flask and Docker in DevOps.

1. Set up Flask Application

Create a new folder for your project and inside it, create the following files.

- ***app.py***

This is your Flask application:

python

Copy code ##

```
from flask import Flask, render_template, request
app = Flask(__name__)
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        event = request.form['event']
        return render_template('success.html', name=name, event=event)
    return render_template('index.html')
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5050)
```

- **templates/index.html**

This is the HTML file that will be served: html

Copy code##

```
<!DOCTYPE html>
<html>
<head>
    <title>Register</title>
</head>
<body>
    <h1>Event Registration</h1>
    <form method="POST">
        <label>Name: <input type="text" name="name" required></label><br>
        <label>Email: <input type="email" name="email" required></label><br>
        <label>Event:
            <select name="event" required>
                <option value="Tech Conference">Tech Conference</option>
                <option value="AI Summit">AI Summit</option>
            </select>
        </label><br>
        <button type="submit">Register</button>
```

```
    </form>
</body>
</html>
```

- **templates/success.html**

```
<!DOCTYPE html>
<html>
<head>
    <title>Success</title>
</head>
<body>
    <h1>Success!</h1>
    <p>{{ name }}, you registered for {{ event }}.</p>
    <a href="/">Back</a>
</body>
</html>
```

2. Create a Dockerfile

The Dockerfile will specify how to build the Docker image for your Flask application:

Dockerfile

dockerfile

Copy code

```
# Use a base Python image
FROM python:3.9-slim
WORKDIR /app
COPY . /app
RUN pip install flask
EXPOSE 5000
CMD ["python", "app.py"]
```

3. Build and Run the Docker Container

Open a terminal in your project folder and run the following commands:
Build the Docker image:
docker build -t mini-event .
Run the container:
docker run -p 5000:5000 mini-event

4. Access the Application

Once the container is running, open a web browser and go to http://localhost:5000.

VIVA QUESTIONS

1. Define HTML.
2. What is Docker?
3. What is Flask?

Experiment: 2

Explore Git and GitHub commands

Aim: To explore Git and GitHub commands
Description: Git and GitHub are two of the most popular tools used for version control and collaboration in software development.

Here are some common Git and GitHub commands

1. **git –version:** Used to show the current version of Git.
2. **mkdir:** Create a directory if not created initially.
   mkdir <file name>
3. **cd:** To go inside the directory and work on its contents.
   The cd command (change directory) is used to navigate into an existing directory.
   cd <file name>
4. **git init:** Initializes a new Git repository in the current directory.
   Analogy: Imagine setting up a new workspace in a physical room and designating it as your project workspace.
5. **git config:** This command sets the name and email address respectively to be used with your commits.
   a. **git config –global user.name "[name]"**
   b. **git config –global user.email "[email address]"**
   Analogy: After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins.
6. **git clone:** Creates a copy of a remote repository on your local machine.
   Analogy: It's like making a photocopy of a book from the library to read and make notes on at your own desk.
   git clone <repository-url>
7. **clear:** For clearing the screen
8. **git status:** This command lists all the files that have to be committed.
   Analogy: The command provides the current working branch. If the files are in the staging area, but not committed, it will be shown by the git status.
9. **git help config:** Take help from the Git help section for different commands and other errors.
   Analogy: A page opens synopsis
   git help config (OR) git config –h
10. **git rm:** This command deletes the file from your working directory and stages the deletion.
    git rm [file]
11. **git remote add origin:** These commands make a bookmark which signifies that this particular remote refers to this URL. This remote will be used to pull any content from the directory and push our local content to the global server.
    git remote add origin https://github.com/kalpanachauhan/test_demo1.git
12. **ls:** To see directories and files in the current directory.
13. **git add:** Stages changes in a file to be ready for commit.
    Analogy: Think of adding items to a shopping cart before you proceed to the checkout.

git add <file-name>

14. **git commit:** Records staged changes with a descriptive message.
    Analogy: To commit our changes(taking a snapshot) and provide a message to remember for future reference.
    git commit -m "Message"
15. **git pull:** Fetches changes from a remote repository and merges them into your current branch.
    Analogy: Imagine updating your notes with the latest information from a shared group document.
16. **git push:** To push all the contents of our local repository that belong to the master branch to the server(Global repository).Uploads your local commits to a remote repository.
    Analogy: Just like sharing your finished project with your team by placing it in a shared folder.
    git push -u origin master
17. **git branch:** Lists all local branches and highlights the current one.
    Analogy: Think of different project directions as separate rooms, and you're checking which room you're currently in.
18. **git merge:** Combines changes from another branch into the current branch.
    Analogy: Similar to merging two streams of thought or work into one coherent piece.
19. **git log:** Lists all local branches and highlights the current one.
    Analogy: Think of different project directions as separate rooms, and you're checking which room you're currently in.
20. **git reset:** Unstages changes in a file, reverting it to the last commit state.
    Analogy: Like removing items from your shopping cart before finalizing your purchase.
21. **git checkout:** Undoing changes
    Analogy: To Blow away all changes since the last commit of the file.
    git checkout<filename>

These are just a few of the many Git and GitHub commands available. There are many other Git commands and functionalities that you can explore to suit your needs.

VIVA QUESTIONS

1. Define Git.
2. What is Github.
3. Difference between Git & Github

Experiment: 3

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

Aim: To practice source code management on GitHub.

Description: To practice source code management on GitHub, you can follow these steps:

- Create a GitHub account if you don't already have one.
- Create a new repository on GitHub.
- Clone the repository to your local machine: $ git clone <repository-url>
- Move to the repository directory: $ cd<repository-url>
- Create a new file in the repository and add the source code written in exercise 1.
- Stage the changes: $ git add<file-name>
- Commit the changes: $ git commit -m "Added source code for a simple user registration form"
- Push the changes to the remote repository: $ git push origin master.
- Verify that the changes are reflected in the repository on GitHub.

These steps demonstrate how to use GitHub for source code management. You can use the same steps to manage any source code projects on GitHub. Additionally, you can also explore GitHub features such as pull requests, code review, and branch management to enhance your source code management workflow.

VIVA QUESTIONS

1. What are the features of Git and Github?
2. Define source code management?
3. Brief description on pull requests, code review and branch management.