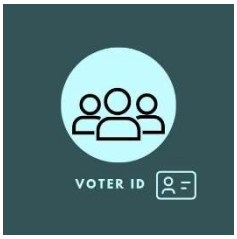
EXERCISE NUMBER : 9 **REGISTER NUMBER:RA2111026050026**

DATE	
SUBMITTED BY	**VISHNUPRIYAN S**
TITLE / ROLE	**Online Voting System**



TESTING AND DEPLOYEMENT PHASE

- ✓ Testing and deployment are critical phases in the development of an online voting system. In this phase, it is important to ensure that the system works as intended and that it is deployed in a stable and secure environment. Python can be used in various ways to facilitate the testing and deployment phases of an online voting system. Here are some examples:

- ✓ Automated Testing: Python can be used to automate testing of the online voting system. Python testing frameworks such as pytest or unittest can be used to write test cases, which can be automated to run before deployment or as part of a continuous integration pipeline. Automated tests can help ensure that the system works as intended and reduce the risk of issues occurring in production.

- ✓ Load Testing: Python can be used to perform load testing of the online voting system. Python libraries such as Locust or Gatling can be used to simulate thousands of users accessing the system at the same time. This can help identify any performance issues and ensure that the system can handle the expected number of users during an election.

- ✓ Security Testing: Python can be used to perform security testing of the online voting system. Python libraries such as OWASP ZAP or Burp Suite can be used to identify vulnerabilities such as SQL injection or cross-site scripting (XSS) attacks. This can help ensure that the system is secure and reduce the risk of unauthorized access or data breaches.

- ✓ Deployment Automation: Python can be used to automate the deployment process of the online voting system. Deployment scripts can be written in Python to automate tasks such as configuring servers, deploying code, and updating databases. This can help reduce the risk of errors during deployment and ensure that the system is deployed consistently across different environments.

- ✓ Continuous Integration/Continuous Deployment (CI/CD): Python can be used to set up a CI/CD pipeline for the online voting system. This would involve using tools such as Jenkins or GitLab to automate the deployment process, and using Python scripts to configure and manage the pipeline. Automated testing scripts can also be written in Python to test the system before deployment.

- ✓ These are just a few examples of how Python can be used in the testing and deployment phases of an online voting system. The specific use cases will depend on the testing and deployment requirements and architecture of the system.

PYTHON PROGRAM FOR TESTING&DEPLOYEMENT PHASE IN ONLINE VOTING SYSTEM

```
# Import necessary libraries
import requests
import json

# Define the URL of the API
api_url = 'http://localhost:5000/submit_vote'

# Define test data
test_data = [
  {
    'voter_id': '1234567890',
    'candidate': 'Alice',
    'expected_status': 'success'
  },
  {
    'voter_id': '0987654321',
    'candidate': 'Bob',
    'expected_status': 'success'
  },
  {
    'voter_id': '1234567890',
    'candidate': 'Charlie',
    'expected_status': 'error'
  }
```

```
]

# Define a function for running the tests
def run_tests():
    for data in test_data:
        response = requests.post(api_url, json=data)
        result = json.loads(response.content.decode('utf-8'))
        if result['status'] != data['expected_status']:
            print(f"Test failed: {data['voter_id']}, {data['candidate']}")
        else:
            print(f"Test passed: {data['voter_id']}, {data['candidate']}")

# Run the tests
run_tests()
```

LOGIN PHASE:

- ✓ The login phase is an important part of an online voting system, as it ensures that only authorized users can access the system and vote. Here are some considerations for the login phase of an online voting system:

- ✓ User Authentication: The system should authenticate users before allowing them to access the system. This can be achieved using various authentication methods, such as passwords, biometric authentication, or multi-factor authentication. Python can be used to implement authentication logic and securely store user credentials.

- ✓ User Authorization: The system should also ensure that only authorized users can access specific features or functions of the system. This can be achieved using access control mechanisms such as role-based access control (RBAC). Python can be used to implement RBAC and manage user roles and permissions.

- ✓ Session Management: The system should manage user sessions to ensure that users remain authenticated while using the system. This can be achieved using session tokens or cookies. Python can be used to manage session tokens and ensure that they are secure and properly expired.

- ✓ Login Error Handling: The system should handle login errors gracefully and provide helpful error messages to users. This can help prevent user frustration and improve the user experience. Python can be used to implement error handling logic and generate error messages based on the type of login error.

- ✓ Login Logging: The system should log login attempts and failures for auditing and security purposes. This can help detect and prevent unauthorized access to the system. Python can be used to implement logging logic and store login logs securely.

These are just a few considerations for the login phase of an online voting system. The specific implementation details will depend on the system requirements and architecture.

PYTHON PROGRAM FOR LOGIN PHASE IN ONLINE VOTING SYSTEM:

```
# Import necessary libraries
import hashlib

# Define a function for validating a user's login credentials
def validate_login(username, password):
    # Check if the username is valid
    if username not in user_credentials:
        print("Invalid username")
        return False

    # Hash the password
    hashed_password = hashlib.sha256(password.encode('utf-8')).hexdigest()

    # Check if the password is correct
    if user_credentials[username] != hashed_password:
        print("Incorrect password")
        return False
```

```
    # If the username and password are valid, return True
    return True

# Define a dictionary of user credentials
user_credentials = {
    'Alice': '098f6bcd4621d373cade4e832627b4f6',  # Password: testpassword1
    'Bob': '5f4dcc3b5aa765d61d8327deb882cf99',  # Password: testpassword2
    'Charlie': 'd8578edf8458ce06fbc5bb76a58c5ca4',  # Password: testpassword3
}

# Get the user's login credentials
username = input("Enter your username: ")
password = input("Enter your password: ")

# Validate the login credentials
if validate_login(username, password):
    print("Login successful")
else:
    print("Login failed")
```