TP Guidé : Selenium et CI/CD - Automatisation des Tests Web

Partie 1 : Mise en place du projet

1.1 Création du projet de base

- Créer un nouveau repository GitHub selenium-cicd-tp

- Structure du projet

```
selenium-cicd-tp/
├── src/
│   ├── index.html
│   ├── style.css
│   └── script.js
├── tests/
│   ├── test_selenium.py
│   └── requirements.txt
├── .github/
│   └── workflows/
│       └── ci-cd.yml
└── README.md
```

- Créer une application web simple
  ```html
  <!DOCTYPE html>
  <html lang="fr">
  <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Calculatrice Simple</title>
      <link rel="stylesheet" href="style.css">
  </head>
  <body>
      <div class="container">
          <h1>Calculatrice Simple</h1>
          <form id="calculator">
              <input type="number" id="num1" placeholder="Premier nombre" required>
              <select id="operation">
                  <option value="add">Addition (+)</option>
                  <option value="subtract">Soustraction (-)</option>
                  <option value="multiply">Multiplication (×)</option>
                  <option value="divide">Division (÷)</option>
              </select>
              <input type="number" id="num2" placeholder="Deuxième nombre" required>
              <button type="submit" id="calculate">Calculer</button>
          </form>
          <div id="result"></div>
      </div>
      <script src="script.js"></script>
  </body>
  </html>
  ```

- CSS basique
  ```css
  .container {
      max-width: 400px;
      margin: 50px auto;
      padding: 20px;
      border: 1px solid #ddd;
      border-radius: 8px;
  }

  input, select, button {
      display: block;
      width: 100%;
      margin: 10px 0;
      padding: 10px;
      font-size: 16px;
  }
  ```

```
#result {
    margin-top: 20px;
    padding: 10px;
    background-color: #f0f0f0;
    border-radius: 4px;
    min-height: 20px;
}
```

- JavaScript fonctionnel
```javascript
document.getElementById('calculator').addEventListener('submit', function(e) {
    e.preventDefault();

    const num1 = parseFloat(document.getElementById('num1').value);
    const num2 = parseFloat(document.getElementById('num2').value);
    const operation = document.getElementById('operation').value;

    let result;

    switch(operation) {
        case 'add':
            result = num1 + num2;
            break;
        case 'subtract':
            result = num1 - num2;
            break;
        case 'multiply':
            result = num1 * num2;
            break;
        case 'divide':
            result = num2 !== 0 ? num1 / num2 : 'Erreur: Division par zéro';
            break;
    }

    document.getElementById('result').textContent = `Résultat: ${result}`;
});
```

Partie 2 : Configuration de Selenium

2.1 Installation des dépendances

- Créer le fichier requirements.txt
  ```
  selenium==4.15.0
  pytest==7.4.3
  pytest-html==4.1.1
  webdriver-manager==4.0.1
  ```

- Installation des versions adéquates
  ```bash
  pip install -r requirements.txt
  ```

2.2 Écriture des tests Selenium

-Tests de base (test_selenium.py)
  ```python
  import pytest
  import time
  from selenium import webdriver
  from selenium.webdriver.common.by import By
  from selenium.webdriver.support.ui import WebDriverWait
  from selenium.webdriver.support import expected_conditions as EC
  from selenium.webdriver.support.ui import Select
  from selenium.webdriver.chrome.options import Options
  from webdriver_manager.chrome import ChromeDriverManager
  from selenium.webdriver.chrome.service import Service
  import os

  class TestCalculator:

      @pytest.fixture(scope="class")
      def driver(self):
          """Configuration du driver Chrome pour les tests"""
          chrome_options = Options()

          # Configuration pour environnement CI/CD
          if os.getenv('CI'):
              chrome_options.add_argument('--headless')
              chrome_options.add_argument('--no-sandbox')
              chrome_options.add_argument('--disable-dev-shm-usage')
              chrome_options.add_argument('--disable-gpu')
              chrome_options.add_argument('--window-size=1920,1080')
```

```
    service = Service(ChromeDriverManager().install())
    driver = webdriver.Chrome(service=service, options=chrome_options)
    driver.implicitly_wait(10)

    yield driver
    driver.quit()

def test_page_loads(self, driver):
    """Test 1: Vérifier que la page se charge correctement"""
    file_path = os.path.abspath("../src/index.html")
    driver.get(f"file://{file_path}")

    # Vérifier le titre
    assert "Calculatrice Simple" in driver.title

    # Vérifier la présence des éléments principaux
    assert driver.find_element(By.ID, "num1").is_displayed()
    assert driver.find_element(By.ID, "num2").is_displayed()
    assert driver.find_element(By.ID, "operation").is_displayed()
    assert driver.find_element(By.ID, "calculate").is_displayed()

def test_addition(self, driver):
    """Test 2: Tester l'addition"""
    file_path = os.path.abspath("../src/index.html")
    driver.get(f"file://{file_path}")

    # Saisir les valeurs
    driver.find_element(By.ID, "num1").send_keys("10")
    driver.find_element(By.ID, "num2").send_keys("5")

    # Sélectionner l'addition
    select = Select(driver.find_element(By.ID, "operation"))
    select.select_by_value("add")

    # Cliquer sur calculer
    driver.find_element(By.ID, "calculate").click()

    # Vérifier le résultat
    result = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "result"))
    )
    assert "Résultat: 15" in result.text

def test_division_by_zero(self, driver):
    """Test 3: Tester la division par zéro"""
    file_path = os.path.abspath("../src/index.html")
    driver.get(f"file://{file_path}")
```

```
        # Saisir les valeurs
        driver.find_element(By.ID, "num1").clear()
        driver.find_element(By.ID, "num1").send_keys("10")
        driver.find_element(By.ID, "num2").clear()
        driver.find_element(By.ID, "num2").send_keys("0")

        # Sélectionner la division
        select = Select(driver.find_element(By.ID, "operation"))
        select.select_by_value("divide")

        driver.find_element(By.ID, "calculate").click()

        # Vérifier le message d'erreur
        result = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.ID, "result"))
        )
        assert "Erreur: Division par zéro" in result.text

    def test_all_operations(self, driver):
        """Test 4: Tester toutes les opérations"""
        file_path = os.path.abspath("../src/index.html")
        driver.get(f"file://{file_path}")

        operations = [
            ("add", "8", "2", "10"),
            ("subtract", "8", "2", "6"),
            ("multiply", "8", "2", "16"),
            ("divide", "8", "2", "4")
        ]

        for op, num1, num2, expected in operations:
            # Nettoyer les champs
            driver.find_element(By.ID, "num1").clear()
            driver.find_element(By.ID, "num2").clear()

            # Saisir les valeurs
            driver.find_element(By.ID, "num1").send_keys(num1)
            driver.find_element(By.ID, "num2").send_keys(num2)

            # Sélectionner l'opération
            select = Select(driver.find_element(By.ID, "operation"))
            select.select_by_value(op)

            # Calculer
            driver.find_element(By.ID, "calculate").click()

            # Vérifier le résultat
```

```
            result = WebDriverWait(driver, 10).until(
                EC.presence_of_element_located((By.ID, "result"))
            )
            assert f"Résultat: {expected}" in result.text

            time.sleep(1)

    if __name__ == "__main__":
        pytest.main(["-v", "--html=report.html", "--self-contained-html"])
    ```
```

2.3 Test en local

- Exécuter les tests
  ```bash
  python -m pytest test_selenium.py -v --html=report.html --self-contained-html
  ```

- Analyser le rapport généré(`report.html`)

  ```

Partie 3 : Configuration CI/CD avec GitHub Actions

3.1 Workflow GitHub Actions

- Créer le fichier workflow (`.github/workflows/ci-cd.yml`)
  ```yaml
  name: CI/CD Pipeline with Selenium Tests

  on:
    push:
      branches: [ main, develop ]
    pull_request:
      branches: [ main ]

  jobs:
    test:
      runs-on: ubuntu-latest

      steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.9'

      - name: Install Chrome
        run: |
          sudo apt-get update
          sudo apt-get install -y google-chrome-stable

      - name: Install dependencies
        run: |
          cd tests
          pip install -r requirements.txt

      - name: Run Selenium Tests
        env:
          CI: true
        run: |
          cd tests
          python -m pytest test_selenium.py -v --html=../test-report.html --self-contained-html

      - name: Upload test results
        uses: actions/upload-artifact@v3
        if: always()
  ```

```
        with:
          name: test-results
          path: test-report.html

    deploy:
      needs: test
      runs-on: ubuntu-latest
      if: github.ref == 'refs/heads/main' && github.event_name == 'push'

      steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Deploy to GitHub Pages
        uses: peaceiris/actions-gh-pages@v3
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./src
```

3.2 Configuration des branches

- Créer une branche develop/dev et pousser votre travail dessus

- Créer une Pull Request vers main

Partie 4 : Métriques et qualité

4.1 Ajout de métriques de couverture

- Modifier requirements.txt
  ```
  selenium==4.15.0
  pytest==7.4.3
  pytest-html==4.1.1
  pytest-cov==4.0.0
  webdriver-manager==4.0.1
  ```

- Configuration pytest(`tests/pytest.ini`)
  ```ini
  [tool:pytest]
  testpaths = .
  python_files = test_*.py
  python_classes = Test*
  ```

```
python_functions = test_*
addopts =
    -v
    --html=report.html
    --self-contained-html
    --cov=../src
    --cov-report=html
    --cov-report=term-missing
```

4.2 Tests de performance

- Ajouter des tests de performance (ajouter à `test_selenium.py`)

```python
def test_page_load_time(self, driver):
    """Test 5: Mesurer le temps de chargement de la page"""
    start_time = time.time()

    file_path = os.path.abspath("../src/index.html")
    driver.get(f"file://{file_path}")

    # Attendre que la page soit complètement chargée
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "calculator"))
    )

    load_time = time.time() - start_time
    print(f"Temps de chargement: {load_time:.2f} secondes")

    # Vérifier que le chargement prend moins de 3 secondes
    assert load_time < 3.0, f"Page trop lente à charger: {load_time:.2f}s"
```

Partie 5 : Exercices pratiques

5.1 : Étendre les tests
Ajoutez les tests suivants :
- Test avec des nombres décimaux
- Test avec des nombres négatifs
- Test de l'interface utilisateur (couleurs, tailles)

5.2 : Page Object Pattern
Refactorisez les tests en utilisant le pattern Page Object : `tests/calculator_page.py`

```python
class CalculatorPage:
    def __init__(self, driver):
        self.driver = driver

    def load_page(self):
        file_path = os.path.abspath("../src/index.html")
        self.driver.get(f"file://{file_path}")

    def enter_first_number(self, value):
        self.driver.find_element(By.ID, "num1").send_keys(str(value))

    def enter_second_number(self, value):
        self.driver.find_element(By.ID, "num2").send_keys(str(value))

    def select_operation(self, operation):
        select = Select(self.driver.find_element(By.ID, "operation"))
        select.select_by_value(operation)

    def click_calculate(self):
        self.driver.find_element(By.ID, "calculate").click()

    def get_result(self):
        result = WebDriverWait(self.driver, 10).until(
            EC.presence_of_element_located((By.ID, "result"))
        )
        return result.text
```

5.3 : Pipeline avancé
Modifiez le workflow pour :
- Exécuter les tests sur plusieurs navigateurs (Chrome, Firefox)
- Ajouter une étape de notification Slack/Email en cas d'échec
- Implémenter un déploiement conditionnel basé sur les tags

---

1. **Avantages observés** :
   - Quels sont les avantages de l'automatisation des tests que vous avez constatés ?
   - Comment le CI/CD améliore-t-il la qualité du code ?

2. **Défis rencontrés** :
   - Quelles difficultés avez-vous rencontrées avec Selenium ?
   - Comment pourriez-vous améliorer la stabilité des tests ?

3. **Métriques** :
   - Quelles métriques sont les plus importantes pour votre projet ?
   - Comment mesurer l'efficacité de votre pipeline CI/CD ?

Ressources supplémentaires

- [Documentation Selenium](https://selenium-python.readthedocs.io/)
- [GitHub Actions Documentation](https://docs.github.com/en/actions)
- [Best Practices pour les tests Selenium](https://www.selenium.dev/documentation/test_practices/)