Integration Testing

Selenium Test
install selenium dependency

```
pip install selenium
```

Download the Browser Web Driver
https://sites.google.com/chromium.org/driver/downloads
select version and download the zip file
if you are using windows then paster driver in c:/drivers folder
for mac/linux : /usr/local/bin or /usr/bin folder

if you don't want to use default location then use below steps

```
from selenium.webdriver.chrome.service import Service

service = Service("C:\Drivers\chromedriver-win64\chromedriver.exe")
driver = webdriver.Chrome(service=service)
```

any one of above option you can use.

Setup Cypress
- Install cypress
- initialize Project
- write your first test case
- run the test in cypress runner or CLI

Set up a Project
for that you must have node js installed in your system. check node version

```
node -v
npm -v
```

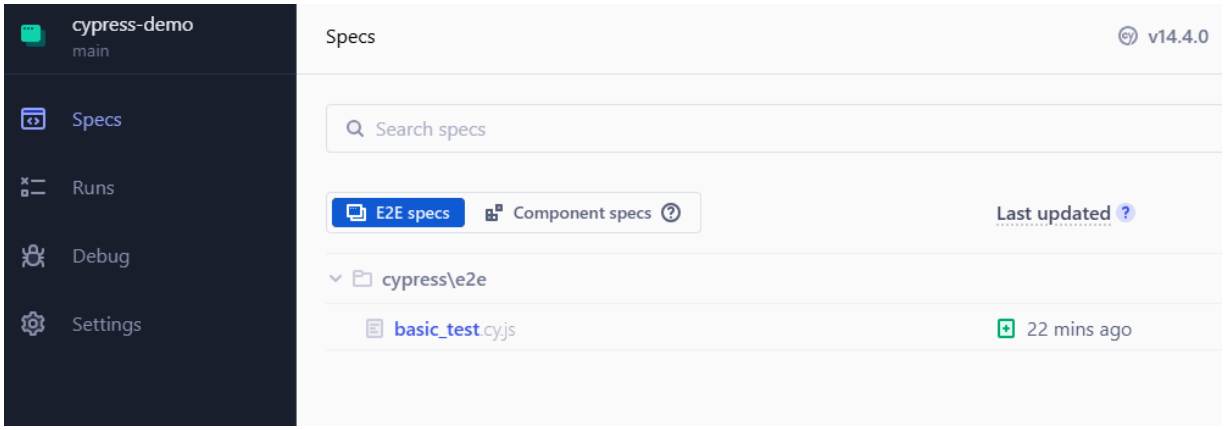
Create Folder cypress-demo
move to the folder cd cypress-demo
initialize project: npm init -y (this will create package.json file)

Install Cypress: npm install cypress --save-dev
after this we will launch cypress: npx cypress open
this command will
create cypress folder
create its config file (cypress.json)
see the test Runner UI

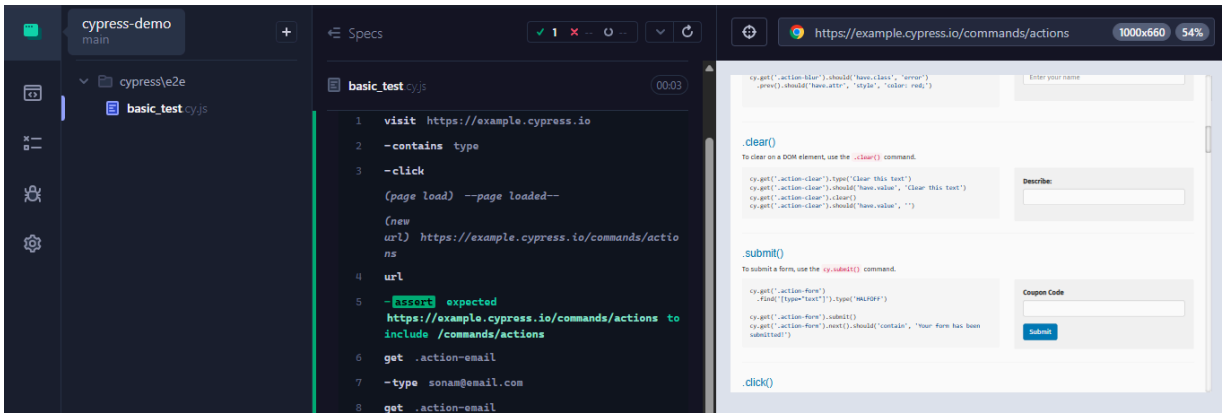
select e2e and then wait it will create folder structure in your project
create folder e2e under cypress folder and write first basic test case
basic_test.cy.js

```
// Test-suite
describe('My First Test', () => {
//Test-Case
it('clicks the link "type"', () => {
cy.visit('https://example.cypress.io')

cy.contains('type').click()

// Should be on a new URL which
// includes '/commands/actions'
cy.url().should('include', '/commands/actions')

// Get an input, type into it
cy.get('.action-email').type('sonam@email.com')

// Verify that the value has been updated
cy.get('.action-email').should('have.value', 'sonam@email.com')
})
})
```

You ca see the below result on cypress Dashboard



click on basic_test



You can see the each step result here.

Let's Setup Code Coverage

pytest-cov is a plugin for pytest which measures code coverage of your python code.
It tells you tahta what percentage of your code is tested and which lines weren't executed during testing

-- pip install pytest-cov

create folder code-coverage
create code as below

```
#fb_login.py
from selenium import webdriver

def get_facebook_title():
```

```
driver = webdriver.Chrome()
driver.get('https://www.facebook.com')
title=driver.title
driver.quit()
return title
```

```
#test_fb_login.py
from fb_login import get_facebook_title

def test_facebook_login_title():
assert "Facebook" in get_facebook_title()
```

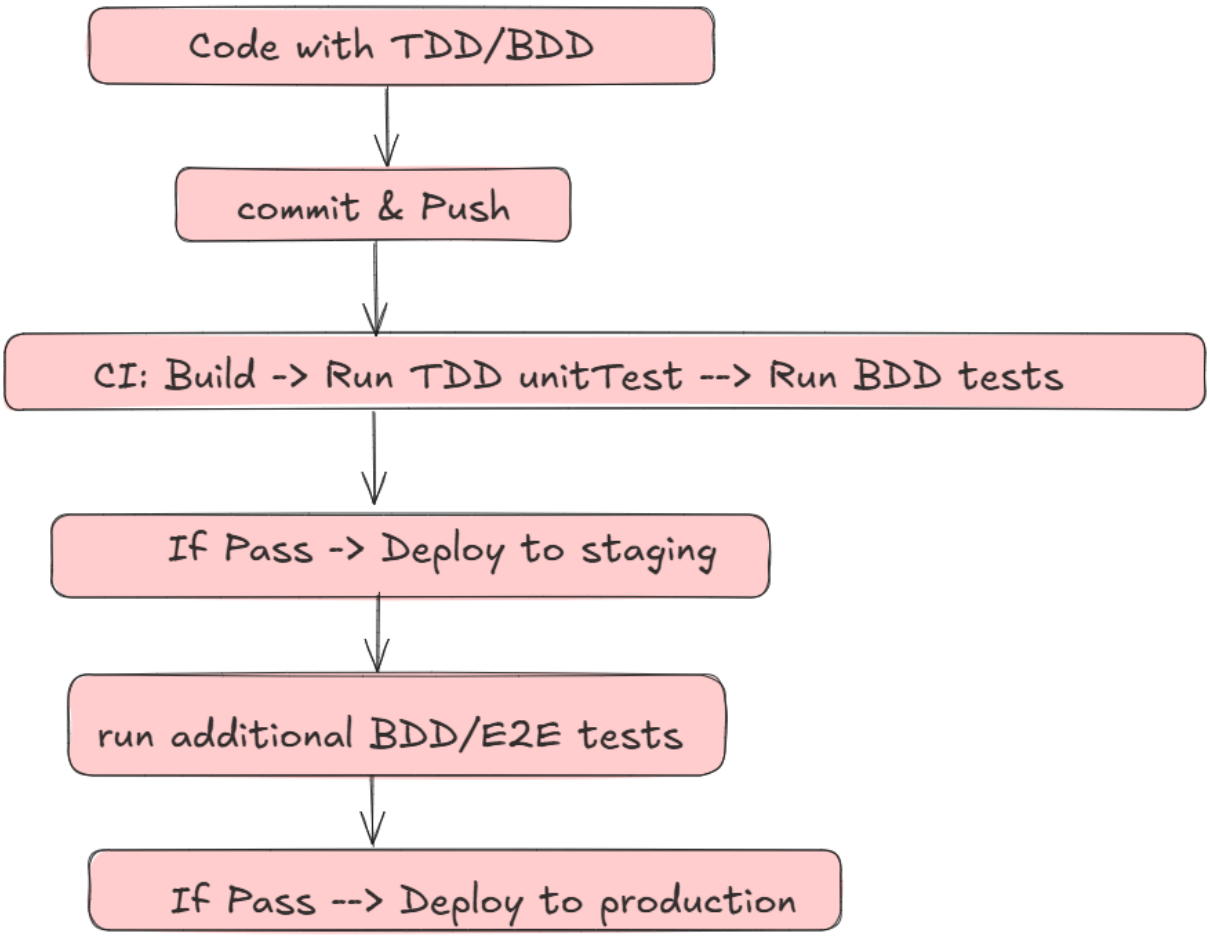
to generate console logging

```
pytest --cov=fb_login

pytest --cov=fb_login test_fb_login.py (for perticular file)
```

for generating HTML DOC

```
pytest --cov=fb_login --cov-report=html
```

How TDD / BDD works with Devops



Let's Setup Github Actions Pipeline to do integration Testing with Node project and Postgres SQL Database.

Create folder for project Node Project
-- mkdir sample-nodeproject
-- cd sample-nodeproject
-- npm init -y (create package.json file)
-- install dependencies:
npm i pg (for postgresql)
npm i jest --save-dev (install testing dependency in development environment)

You can use this reference code: https://github.com/sonam-niit/node-test

open package.json and edit script you need to add just 2 lines of code

```
 "scripts": {
```

```
"test": "jest",
"test:integration": "jest tests/integration"
},
```

also create jest.config.js

```
module.exports = {
testEnvironment: 'node',
testMatch: ['**/tests/**/*.test.js']
};
```

After this for writing test case create folder named tests/integration folder
create file db.test.js

```
const { Client } = require('pg');

describe('Postgres Integration Test', () => {
let client;

beforeAll(async () => {
client = new Client({
connectionString: process.env.DATABASE_URL,
});
await client.connect();
});

afterAll(async () => {
await client.end();
});

test('should connect and return current timestamp', async () => {
const res = await client.query('SELECT NOW()');
expect(res.rowCount).toBe(1);
console.log('Current time from DB:', res.rows[0].now);
});

test('should create and query a test table', async () => {
await client.query('CREATE TABLE IF NOT EXISTS test_table(id SERIAL PRIMARY
KEY, name TEXT)');
await client.query('INSERT INTO test_table(name) VALUES ($1)', ['hello']);
const res = await client.query('SELECT * FROM test_table WHERE name = $1',
['hello']);
```

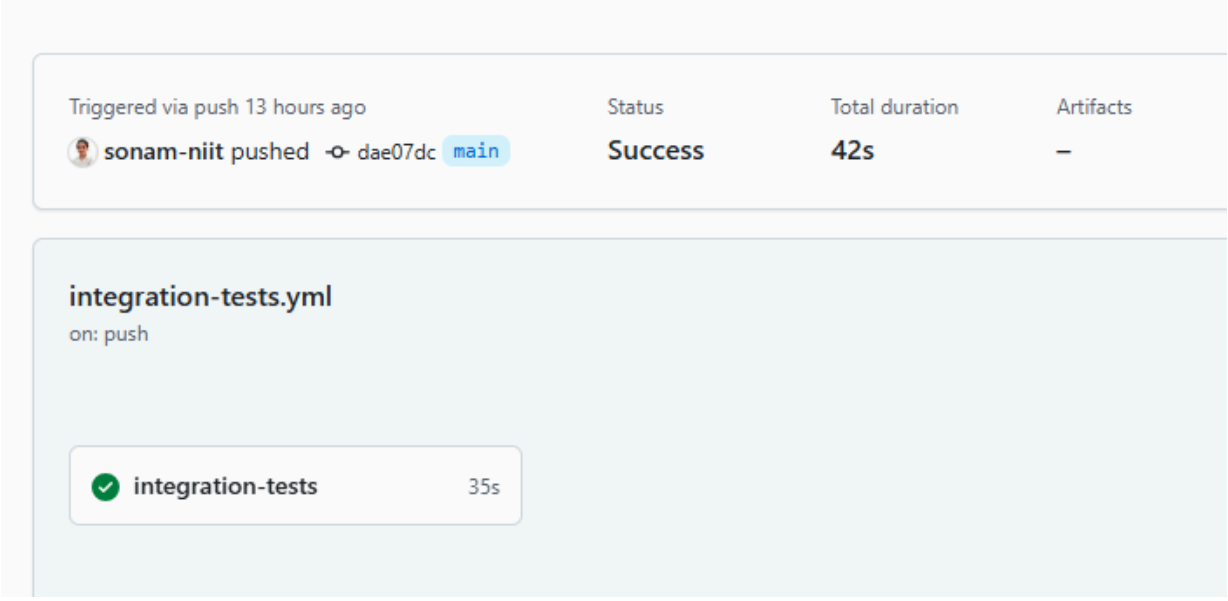
```
expect(res.rows.length).toBeGreaterThan(0);
expect(res.rows[0].name).toBe('hello');
});
});
```

After this to run the test case as CI
create folder .github/workflows
create workflow yml file (integration-tests.yml)

```
name: Integration Tests

on:
push:
branches:
- main
pull_request:

jobs:
integration-tests:
runs-on: ubuntu-latest

services:
postgres:
image: postgres:15
env:
POSTGRES_USER: test_user
POSTGRES_PASSWORD: test_password
POSTGRES_DB: test_db
ports:
- 5432:5432
options: >-
--health-cmd pg_isready
--health-interval 10s
--health-timeout 5s
--health-retries 5

env:
DATABASE_URL: postgres://test_user:test_password@localhost:5432/test_db

steps:
- name: Checkout repository
```

```
uses: actions/checkout@v4

- name: Setup Node.js
uses: actions/setup-node@v4
with:
node-version: 20

- name: Install dependencies
run: npm ci

- name: Wait for Postgres
run: |
until pg_isready -h localhost -p 5432 -U test_user; do
echo "Waiting for postgres..."
sleep 2
done

- name: Run integration tests
env:
DATABASE_URL: ${{ env.DATABASE_URL }}
run: npm run test:integration
```

Now you push this code on Github and check actions
It will download postgres do connectivity pass test case and then destroy created resources.



https://github.com/sonam-niit/node-test/actions/runs/15366091628/job/43239170285
(check this link to see the result)