POM (Page Object Model)

Hoa Doan

12 Aug 2020

Nash Tech.

Agenda

1. What is POM?
2. Why to use POM?
3. How to implement POM?
4. POM Structure
5. Q&A

What is POM?

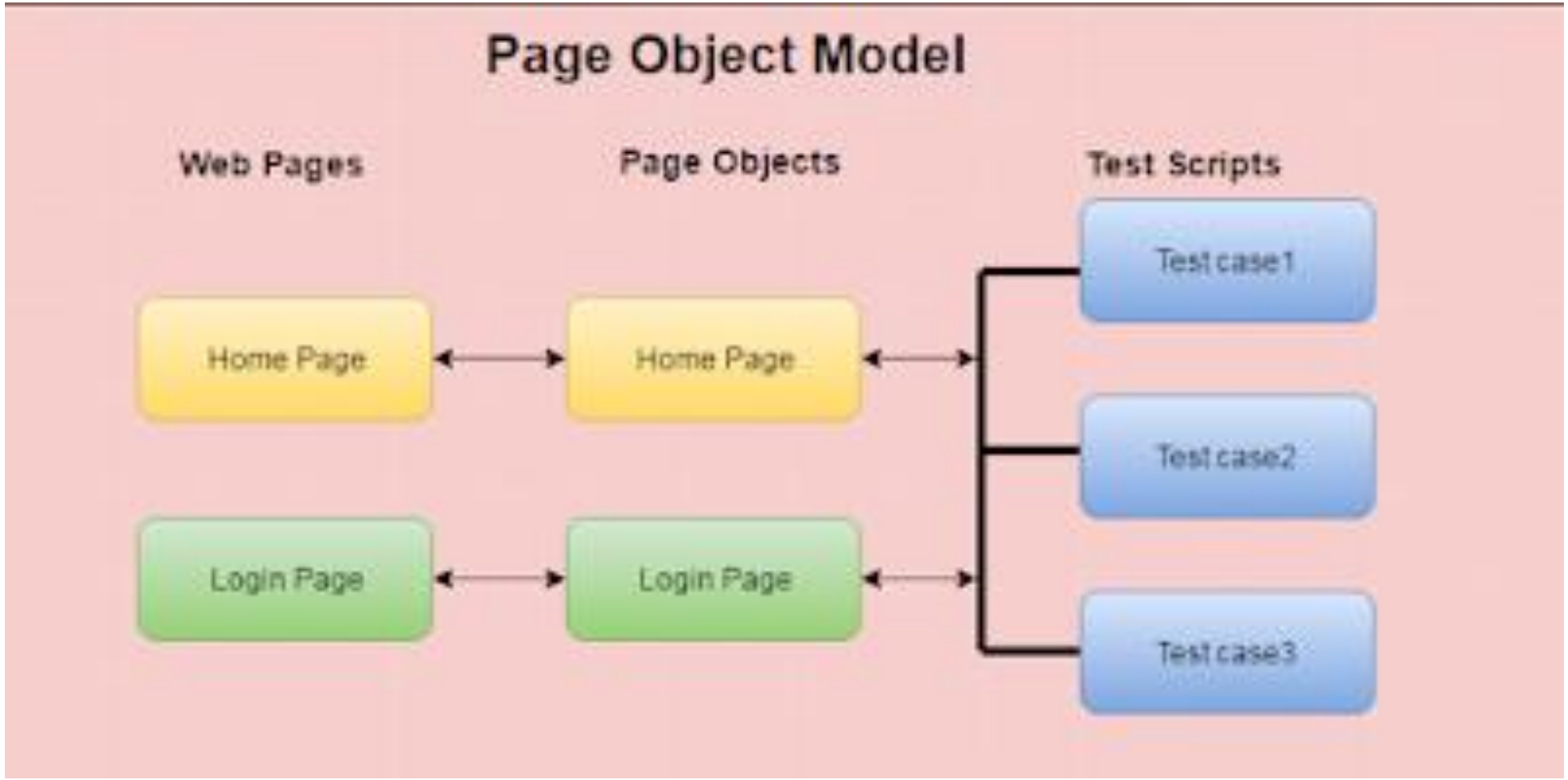
- Test Automation without POM

```
class SampleTestSuite(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.maximize_window()

    def test_login(self):
        self.driver.get(EnvSetup.SITE + cc.URL['LOGIN_URL'])
        self.driver.find_element(By.ID, 'user_email').send_keys('hoadoanthingocvn@gmail.com')
        self.driver.find_element(By.ID, 'user_password').send_keys('12345678')
        self.driver.find_element(By.NAME, 'commit').click()
        is_home_visible = True
        try:
            self.driver.find_element_by_link_text('Home1')
        except NoSuchElementException:
            is_home_visible = False
        assert_that(is_home_visible, equal_to(True), 'Verify home page')

    def tearDown(self):
        self.driver.quit()
```

What is POM?

- Design Pattern.
- Creates an **Object Repository** for storing all web elements.
  - Each web page of an application as a class file.
  - Each class file will contain only corresponding web page elements (locators, methods).

What is POM?


Page Object Model
Web Pages
Page Objects
Test Scripts
Home Page
Home Page
Login Page
Login Page
Test case1
Test case2
Test case3

Why to use POM?

- There is a clean separation between test code and page specific code such as locators (or their use if you're using a UI Map) and layout.
- There is a single repository for the services or operations offered by the page rather than having these services scattered throughout the tests.

➔ **Helps with easy maintenance → reducing errors**
➔ **Helps with reusing code→ save time and effort**
➔ **Readability and Reliability of scripts**

How to implement POM

- **Step 1**: Create a **base** class.
- **Step 2**: Identify web pages and create class (inherit from base class) for each web page.
- **Step 3**: Identify web elements (locators) and methods to interact with web elements.
  - Sequential locators and methods should be followed by UI order.
- **Step 4**: Run and Validate.

Note: Page objects should never make verifications or assertions

POM Structure - Scenario

- Scenario: update password unsuccessfully
- Steps
  1. Go to Change Password page
  2. Input 123456 into Password
  3. Click Change password button
- Expected Result:

  - Verify error message “Password is too short (minimum is 6 characters)” is shown

POM Structure - Example

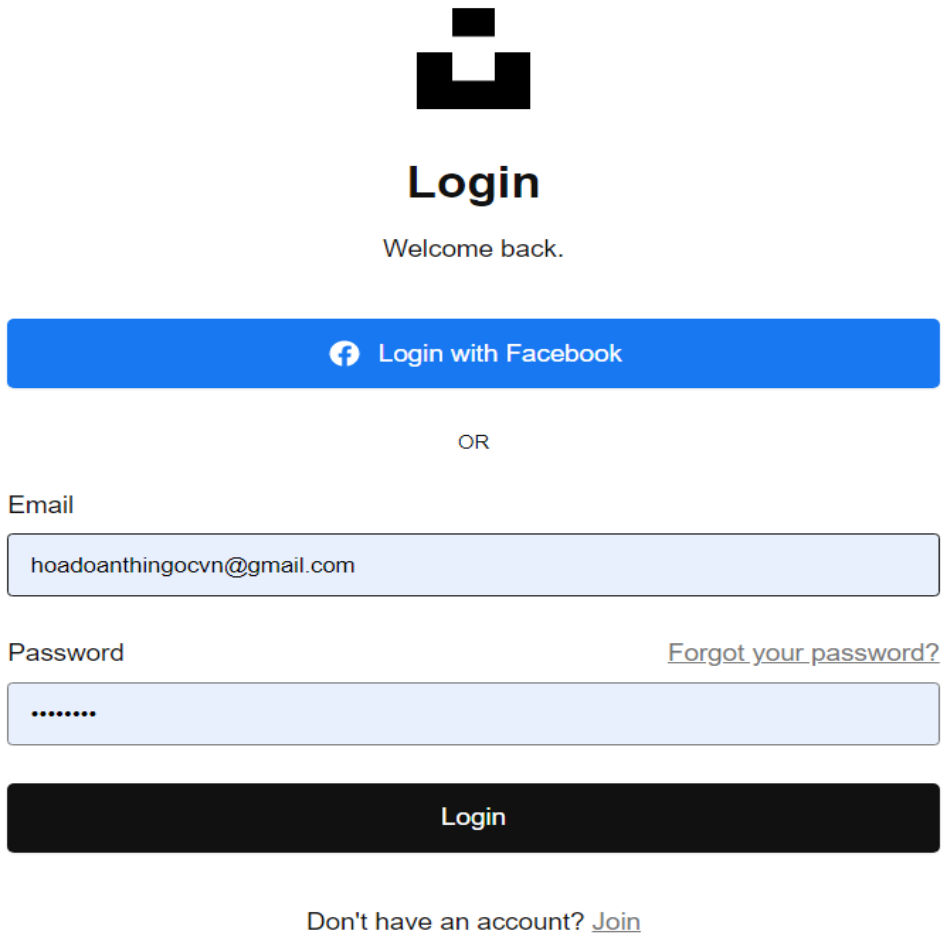
POM Structure – Base class

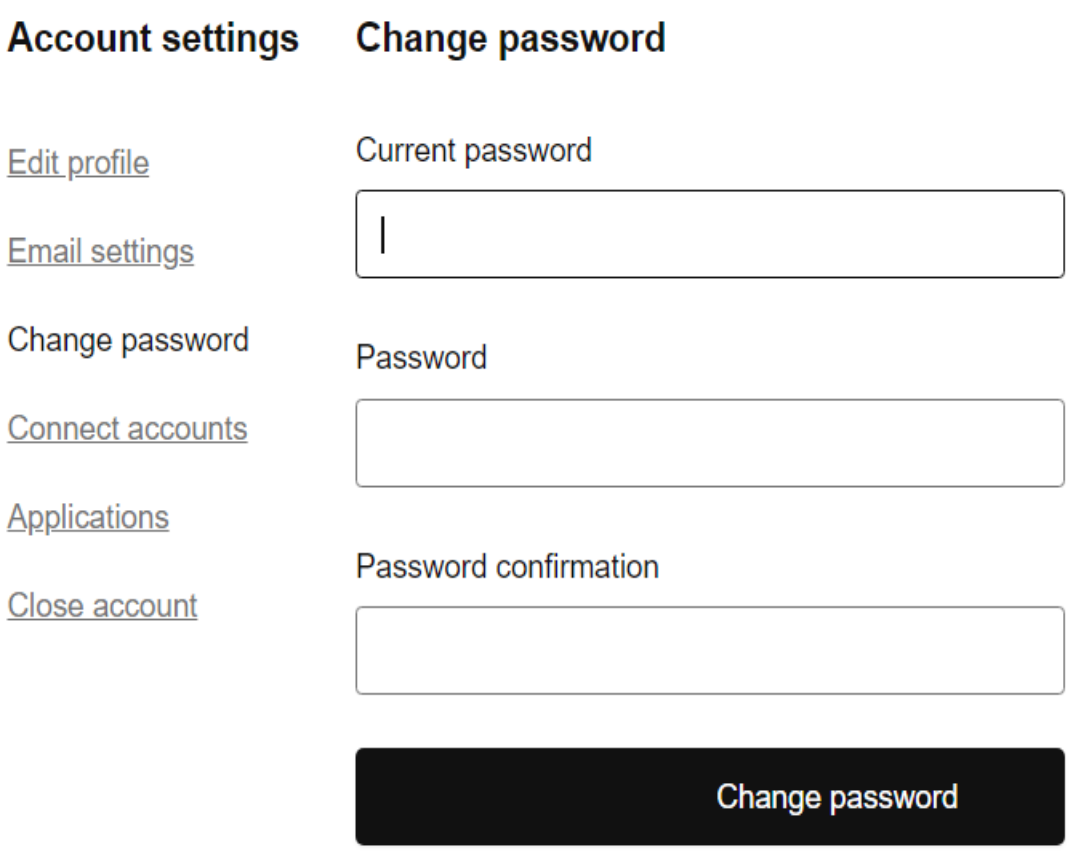
Step 1: Create base class

```python
class BasePage(object):

    def __init__(self, driver):
        self.driver = driver

    def navigate(self, url):
        self.driver.get(url)

    def wait_element_to_be_clickable(self, tuple_locator, timeout=EnvSetup.SELENIUM_TIMEOUT_SECONDS):
        wait = WebDriverWait(self.driver, timeout)
        element = wait.until(EC.element_to_be_clickable(tuple_locator))
        return element

    def wait_element_to_be_visible(self, tuple_locator, timeout=EnvSetup.SELENIUM_TIMEOUT_SECONDS):
        wait = WebDriverWait(self.driver, timeout)
        element = wait.until(EC.visibility_of_element_located(tuple_locator))
        return element

    def wait_elements_to_be_present(self, tuple_locator, timeout=EnvSetup.SELENIUM_TIMEOUT_SECONDS):
        wait = WebDriverWait(self.driver, timeout)
        element = wait.until(EC.presence_of_all_elements_located(tuple_locator))
        return element

    def click_link_name(self, link_name, timeout=EnvSetup.SELENIUM_TIMEOUT_SECONDS):
        element = self.wait_element_to_be_clickable((By.LINK_TEXT, link_name), timeout)
        element.click()

    def click_element(self, tuple_locator, timeout=EnvSetup.SELENIUM_TIMEOUT_SECONDS):
        element = self.wait_element_to_be_clickable(tuple_locator, timeout)
        element.click()

    def type_text(self, tuple_selector, value=None, tab=None, enter=None):
        element = self.wait_element_to_be_visible(tuple_selector)
        if value:
            element.send_keys(value)
        if tab:
            element.send_keys(Keys.TAB)
        if enter:
            element.send_keys(Keys.ENTER)
```

POM Structure – Web Pages

- **Step 2**: Identify web pages and create class for each page





POM Structure – web page class

Step 3: Identify web elements and methods to interact with web elements

```python
class LoginPage(BasePage):

    def input_user_email(self, email):
        locator = constants.LOGIN_PAGE['USER_EMAIL']
        self.type_text(locator, email)

    def input_user_password(self, password):
        locator = constants.LOGIN_PAGE['USER_PASSWORD']
        self.type_text(locator, password)

    def click_login_button(self):
        locator = constants.LOGIN_PAGE['LOGIN_COMMIT_BUTTON']
        self.click_element(locator)

    def login(self, email, password):
        self.input_user_email(email)
        self.input_user_password(password)
        self.click_login_button()
```

```python
class ChangePasswordPage(BasePage):

    def input_current_password(self, current_password):
        locator = constants.CHANGE_PASSWORD['CURRENT_PASSWORD_TEXTBOX']
        self.type_text(locator, current_password)

    def input_password(self, password):
        locator = constants.CHANGE_PASSWORD['PASSWORD_TEXTBOX']
        self.type_text(locator, password)

    def input_password_confirmation(self, password_confirmation):
        locator = constants.CHANGE_PASSWORD['PASSWORD_CONFIRMATION_TEXTBOX']
        self.type_text(locator, password_confirmation)

    def click_change_password_button(self):
        locator = constants.CHANGE_PASSWORD['CHANGE_PASSWORD_BUTTON']
        self.click_element(locator)
```

POM Structure – locators

Step 3: Add locator into separated place

```
LOGIN_PAGE = {
    'USER_EMAIL': (By.ID, 'user_email'),
    'USER_PASSWORD': (By.ID, 'user_password'),
    'LOGIN_COMMIT_BUTTON': (By.NAME, 'commit'),
}

CHANGE_PASSWORD = {
    'CURRENT_PASSWORD_TEXTBOX': (By.ID, 'user_current_password'),
    'PASSWORD_TEXTBOX': (By.ID, 'user_password'),
    'PASSWORD_CONFIRMATION_TEXTBOX': (By.ID, 'user_password_confirmation'),
    'CHANGE_PASSWORD_BUTTON': (By.NAME, 'commit'),
    'ERROR_MESSAGE': (By.CSS_SELECTOR, '.form-error-header ul'),
}
```

Review

- Each page is class.
- No assertion or verification in page objects.
- The sequential locators and methods follow UI order.


Q
&
A

References

- https://www.selenium.dev/documentation/en/guidelines_and_recommendations/page_object_models/
- https://selenium-python.readthedocs.io/page-objects.html
- https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/