



По любым возникающим вопросам вы всегда можете обратиться в наш онлайн-чат, который находится в правом нижнем углу страницы. Удобная база знаний 24/7 и оперативные ответы наших специалистов с 10 до 22 (МСК) ⇓

[Перейти В Базу Знаний FAQ](#)

Курс > Авто... > Моду... > 26.2. ...

26.2. Пример простой реализации PageObject





Давайте теперь посмотрим, как можно применить паттерн *PageObject* на примере страницы авторизации нашего проекта PetFriend. Рекомендуем собственноручно повторить весь код.



В первую очередь мы должны изучить страницу и понять, какие
ней присутствуют тестируемые элементы.



PetFriends

Социальная сеть для любителей животных

Электронная почта

Пароль

Войти

[Зарегистрироваться](#)

На странице присутствуют:

- Форма авторизации с двумя полями:
 1. Электронная почта.
 2. Пароль.
- Кнопка «Войти».
- Ссылка «Зарегистрироваться».

Давайте проверим механизм авторизации. Зайдём с конца и посмотрим, какой код теста мы можем получить.



```
from pages.auth_page import AuthPage
import time
```

```
def test_auth_page(selenium):
```

```
    page = AuthPage(selenium)
    page.enter_email("email@gmail.com")
    page.enter_pass("pass")
    page.btn_click()
```

```
    #знак != или == будет зависеть от того, верные или неверные
    данные мы вводим
```

```
    assert page.get_relative_link() != '/all_pets', "login error"
```

```
    time.sleep(5) #задержка для учебных целей
```

Мы видим, что для написания нашего теста нам не понадобилось думать о конкретных локаторах элементов, об адресе странице, специфичных методах *Selenium*. Код вполне понятен даже человеку без глубоких знаний в *HTML*, *WebDriver* и тому подобном. Заметьте, если мы перенесём наш сайт на новый домен, код этого теста останется рабочим, необходимо будет лишь поменять адрес в одном месте, а не в десятках тестов.

Как же мы добились такой логичности и абстрактности? Всё благодаря *PageObject*!

Перед тем, как мы приступим к коду, рекомендуем изучить документацию к библиотеке *Selenium Python* (с поправкой, что в примерах там приводится фреймворк *unittest*). Сразу оговоримся: в текущем уроке мы самостоятельно воссоздадим все нужные классы для *PageObject*, значительно упростив код для учебных целей. В дальнейшем мы будем использовать готовые библиотеки, которые ещё больше упростят процесс.



Рекомендуем файлы с определением классов страниц помещать в папку `pages` вашего проекта. В ней у нас будут файлы `base_page.py`, `locators.py`, `auth_page.py`.

Давайте создадим базовый класс для некой абстрактной страницы.

```
from urllib.parse import urlparse

class BasePage(object):
    # конструктор класса - специальный метод с ключевым
    # словом __init__
    # Нам нужны объект веб-драйвера, адрес страницы и время
    # ожидания элементов
    def __init__(self, driver, url, timeout=10):
        self.driver = driver
        self.url = url
        self.driver.implicitly_wait(timeout)

    def get_relative_link(self):
        url = urlparse(self.driver.current_url)
        return url.path
```

В нашем классе пока только два метода. Метод `__init__` будет запускаться в момент создания объекта из класса и устанавливать нужные свойства. `driver.implicitly_wait(10)` — задаёт максимальное время ожидания в секундах, в течении которого *WebDriver* пытается найти элемент, если он недоступен сразу же.

Метод `get_relative_link` возвращает относительный путь до текущей страницы (без домена). Он будет полезен для понимания того, на какой странице мы сейчас находимся при редиректах от сервера или при отправке форм при нажатиях кнопки авторизации. При этом мы не хотим зависеть от текущего домена сайта,



дополнительных *get*-параметров или протокола (*http* или *https*) и проверяем только часть *url*, содержащую путь к файлу. Для этого нам необходима библиотека *urllib*.

Если у вас в системе она не установлена, установите её через *pip*:

```
pip install urllib3
```

Также создадим файл с определением локаторов нужных страниц. Они будут объединяться в классы, согласно тому, каким страницам принадлежат.

```
from selenium.webdriver.common.by import By  
  
class AuthLocators:  
    AUTH_EMAIL = (By.ID, "email")  
    AUTH_PASS = (By.ID, "pass")  
    AUTH_BTN = (By.CLASS_NAME, "btn-success")
```

Тут нам необходимо импортировать класс *By*, содержащий определения разных типов поиска. Далее мы внутри класса определяем локаторы в виде списка с типом поиска и поисковой фразой (строкой локатора).

Теперь создадим на её основе класс страницы авторизации:



```
from .base_page import BasePage
from .locators import AuthLocators

import time,os

class AuthPage(BasePage):

    def __init__(self, driver,timeout=10):
        super().__init__(driver, timeout)
        url = os.getenv("LOGIN_URL") or
"http://petfriends1.herokuapp.com/login"
        driver.get(url)
        #создаем нужные элементы
        self.email = driver.find_element(*AuthLocators.AUTH_EMAIL)
        self.password =
driver.find_element(*AuthLocators.AUTH_PASS)
        self.btn = driver.find_element(*AuthLocators.AUTH_BTN)
        time.sleep(3)

    def enter_email(self, value):
        self.email.send_keys(value)

    def enter_pass(self, value):
        self.password.send_keys(value)

    def btn_click(self):
        self.btn.click()
        time.sleep(3) #ждем реакции
```

Обратите внимание на строку:

```
super().__init__(driver, timeout)
```

Тут мы обращаемся к методу `__init__` нашего родительского класса `BasePage`, так как наш новый метод `__init__` затрёт его.



В корне нашего проекта создадим файл `test_auth_page.py` (содержание приведено в начале юнита). Теперь мы можем заметить, что большинство методов объекта `page` мы определили внутри класса страницы авторизации, но один был унаследован от базового класса.

Запускать тест мы будем из командной строки (из директории проекта):

```
python -m pytest -v --driver Chrome --driver-path  
<chromedriver_directory>/<chromedriver_file> test_auth_page.py
```

В случае верного ввода *e-mail* и пароля браузер перейдет на страницу с каталогом животных, и тест будет «зелёным». При неверном вводе данных мы останемся на странице авторизации, и тест будет «красным».

Задание 26.2.1

1/1 point (graded)

Какие классы мы использовали?

- ☒ Базовый класс страницы
- ☐ Базовый класс веб-элементов
- ☒ Класс для страницы авторизации
- ☒ Класс для локаторов
- ☐ Класс для полей ввода



☐ Класс для описания тестов



? **Подсказка (1 из 2):** Для тестов мы используем функциональный подход.

Следующая Подсказка

Подсказка (2 из 2): Для простоты восприятия мы пожертвовали разделением логики страниц и элементов. Посмотрите, как реализована работа с полями ввода и кнопкой у нас.

Отправить

✓ Correct (1/1 point)

Задание 26.2.2

1/1 point (graded)

Как происходит наследование от базового класса?

☐ `class MainPage extends Foo`

☐ `class SecondClass : public BaseClass`

☒ `class Bar(WebPage):` ✓

Отправить



✓ Correct (1/1 point)

© Все права защищены

[Help center](#) [Политика конфиденциальности](#)
[Пользовательское соглашение](#)



SKILLFACTORY

Built on **OPENedX** by **RACCOONGANG** 

