Flask Integration Testing

Contents

Overview

Integration testing is a type of software testing in which we test the application as a whole, rather than mocking the application to it's routes as we do in unit testing.

We will use the Python package `selenium` to simulate a user interacting with our application directly, and test the results are as expected.

Setup

We can use the `LiveServerTestCase` class to create a live instance of our application for our integration tests to use, so we don't need the application to be running for the tests to work.

```
from flask_testing import LiveServerTestCase
```

We must create a subclass of this, and define the following methods:

1. `create_app`: run once, at the very start of testing - here, we overwrite the app's config
2. `setUp`: run before every test case - here, we setup the driver and create our test database
3. `tearDown`: run after every test case - here, we quit the driver and drop the test database

▶ Example

Test Cases

Once we have created a `TestBase` class, which inherits from the `LiveServerTestCase` class, we can define some test cases.

Each test case must be defined within a class which inherits from `TestBase`.

It should now look something like this:

```
class TestBase(LiveServerTestCase):
    def create_app(self):
        ...
        return app

    def setUp(self):
        ...

    def tearDown(self):
        ...

class TestExample(TestBase):
    def test_case_1(self):
        ...
```

XPaths

XPaths are essentially a way to find any element, such as an input field or button, on any HTML or XML document.

Selenium can use an XPath to find the element we want, and we can then manipulate this element in our testing.

▶ Finding the XPath of an element in Chrome

Selenium

We can use the `selenium` driver to find elements on a page, and do *things* with these elements.

We use the following syntax to find an element on the page:

```
element = self.driver.find_element_by_xpath('<XPath>')
```

We can then use any of the following methods on this element:

```
element.click()
element.send_keys('<any string>') # simulates typing
element.clear()
```

We can also inspect the text inside the element using

```
element.text
```

▶ Example

Tutorial

Requirements

An **Ubuntu 18.04** VM with Python installed and port 5000 open.

Note: This is unlikely to work on Ubuntu 20.04.

Installation

Run the following commands to install `chromium-browser` and `chromedriver`:

Installing the browser

```
sudo apt install chromium-browser -y
```

Installing the driver (must have the browser installed for this to work!)

```
sudo apt install wget unzip -y
version=$(curl -s
https://chromedriver.storage.googleapis.com/LATEST_RELEASE_$(chromium-browser --
version | grep -oP 'Chromium \K\d+'))
wget
https://chromedriver.storage.googleapis.com/${version}/chromedriver_linux64.zip
sudo unzip chromedriver_linux64.zip -d /usr/bin
rm chromedriver_linux64.zip
```

Clone down this repository and change directory into it:

```
git clone https://github.com/QACTrainers/selenium-example.git
cd selenium-example
```

Install all `pip` dependencies in a virtual environment:

```
sudo apt install python3 python3-pip python3-venv -y
python3 -m venv venv
source venv/bin/activate
pip3 install -r requirements.txt
```

Running the application

Let's see what the application is doing.

Use

```
python3 create.py
python3 app.py
```

You should be able to see that we can submit entries that will show in the **History** section of the index page.

Submitting an empty entry will give us an error, saying "*The name field can't be empty!*"

Getting the XPath

Let's get the XPath of where the error message should be.

On your app, submit an empty name using the ✓. An error message should pop up as expected.

Follow the tutorial here to get the XPath of this error message.

Writing a test case

Let's create a test case to check the validation of our form, so that we know the user can't submit an empty input.

In `tests/test_int.py, line 62`, configure `test_empty_validation` as follows:

```
    def test_empty_validation(self):
        self.submit_input('')
        self.assertIn(url_for('index'), self.driver.current_url)

        text = self.driver.find_element_by_xpath('<XPath>').text
        self.assertIn("The name field can't be empty!", text)

        entries = Games.query.all()
        self.assertEqual(len(entries), 0) # database should be empty
```

Make sure to replace `'<XPath>'` with the XPath we found in the previous step!
Note: The `self.submit_input` method is a custom-defined method for this demonstration, and is not provided by default.
Note: The `self.submit_input` method is a custom-defined method for this demonstration, and is not provided by default.

We are checking 3 things here:

1. We are redirected back to the index page correctly,
2. The error message is displayed properly,
3. The database is still empty, so the empty entry was ignored as expected.

Running the tests

Run the tests using

```
python3 -m pytest
```

Exercises

Write a test case for `test_length_validation`. If the submitted input has a length greater than 30, the error "*This name is too long!*" should be displayed, and the input should not be added to the database.

Use the `test_empty_validation` method for reference.