QA on Autopilot

Approaches To Web Automation Testing With Selenium

Who are we?

- Engineers working with CBS Interactive's Advanced Technology group
- Primarily working on the central content management system that powers CNet, CBSNews, CBSSports, others...
- Designers and maintainers of a proprietary Selenium framework used for the CMS and other applications at CBSi

S

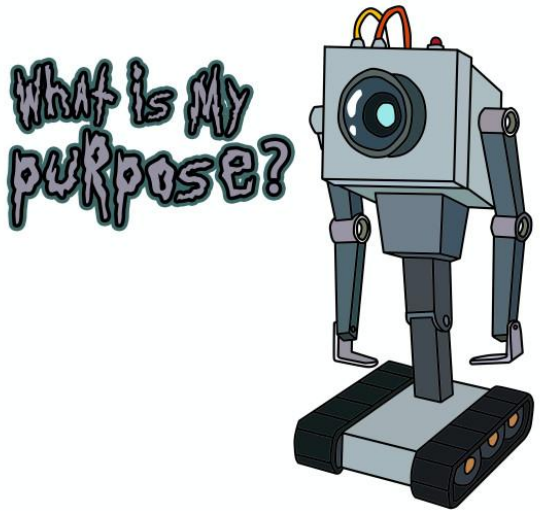
Shoutout to QA

Kshama Shetty
Quality Assurance Ninja

S

Why Automated Testing?

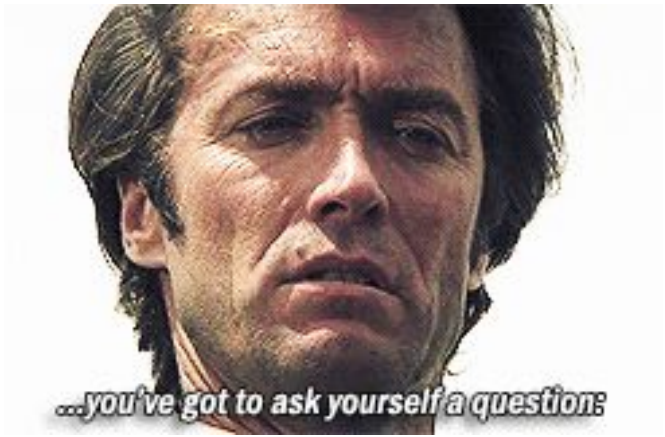
- Streamlines the QA process and establishes consistency
  - Run the same tests every time.
  - Failures can be verified by past test runs.
- Allows humans to focus on human tests
  - Machines can save time by running tests for you.
- Repeatable Regression Testing
  - Easier to test entire features for each new release.



S

Including Automated Testing in a New Project

- Carefully examine what **should** be automated (hint: not every test case!)
  - Questions to ask:
    - If this test were not automated, would I consider it part of a full regression?
    - Is there value in running this test case across future releases of the application?
    - Are the success and failure criteria clearly defined? (Objective vs Subjective)
    - Is the development time for the test worth the trade-off of automation?
    - Is this workflow part of the “critical path”?



D

Introducing AT in a New Project - Continued

- Examples:
  - Long, tedious test cases involving lots of data generation/page clicks with easily identifiable expected outcomes. **(Yes)**
  - Questions of “look and feel”. Is the font too big? Is the spacing right? **(No)**
  - Is the font on input controls exactly 12px? Are the fields in the form in the right order? **(Maybe)**
  - Does performing a procedure no longer cause an error? **(Maybe)**

Introducing AT in a New Project - Continued

- Make automated integration testing part of the development process: “Workflow Driven Development”
  - Development of application features is driven by user stories.
  - As controls/pages are created/populated, model those pages and controls in your testing framework.
  - Convert user stories to clear and concise test case steps

D

Example of WDD

1. User story: “User can populate an article with a headline and body, then save the changes.”
2. App Development: build the controls/page/functions necessary to complete the task - and their corresponding unit tests where applicable!
    - Article page
    - Text input control (headline)
    - contenteditable div control (body)
    - Publish button/action
3. Convert User story to a test case:
    - Navigate to the article page
    - Populate the article headline with text
    - Populate the body with text
    - Click the publish button
    - Assert that the headline/body persisted.

D

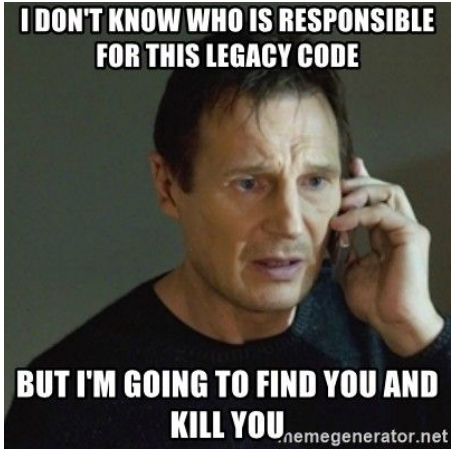
Example of WDD - continued

4. Add the page and controls to the testing framework: Work just like in the app case - controls => pages => functionality
    - Model for text input controls (headline)
    - Model for contenteditable div control (body)
    - Model for button controls (publish button)
    - Class representing article page with properties for headline, body, and publish button.
5. Convert the test case to code steps
6. ...
7. Profit!

D

Introducing Automated Testing To An Existing Project

- Not everything needs to be tested immediately
- Assess the battlefield
  - Does your team have established, repeatable test cases?
  - Is management on board with the introduction of automated testing?
    - Time will need to be spent working on future tests
  - What does your application's DOM look like?
    - Could you reasonably write selectors for testing purposes or is a refactor necessary?



J

Plan of attack

- Consider starting with some of your team's simplest test cases
  - Allows you to focus on establishing the framework your team will use for writing future tests
  - Possible candidates:
    - Login
    - Navigation
- Identify areas in the existing project that would be the biggest QA wins to automate
  - Tests that end up consuming the largest amount of time during your projects QA cycle
  - Tests with a large number of permutations that end up getting skipped
- Future application tickets should plan for integration testing impact

J

Establishing a Selenium Framework

- Selenium has a large number of official and unofficial language bindings
- Your framework should leverage an existing testing framework for gathering and running your tests
  - For instance, Python has: “unittest”, “nose”, “pytest”, etc.
- Engineer your tests like you would your application
  - Peppering dom selectors throughout your tests will create future headache
  - Consolidating common application interactions into functions will help with future updates
    - What if your application’s login requires two-factor authentication in the future?

J

Challenges of Automated Testing

- How do I write automated tests?
- How can what I write be reliable and reusable?
- Should I expect my environment to be in a certain state?
    - Should I expect data to be pre-existing, or should I expect a clean slate?
- Time! (Everyone’s favorite challenge)
    - Am I wasting or saving time?

S

Challenge: Writing “Automatable” Test Cases

S

Find Out What Your QA Does

Have your QA write their tests cases in pseudocode fashion

Human steps:
Testing success on the login page:
1. Open the login page
2. Fill in a username in the username field
3. Fill in the password in the password field
4. Click the login button
5. It should redirect to the homepage

Testing failure on the login page
1. Open the login page
2. Fill in an incorrect username in the username field
3. Fill in the incorrect password in the password field
4. Click the login button
5. It should not redirect to the homepage

Pseudocode:
Testing success on the login page:
1. Open the login page
2. Fill in login form with correct credentials
3. It should redirect to the homepage

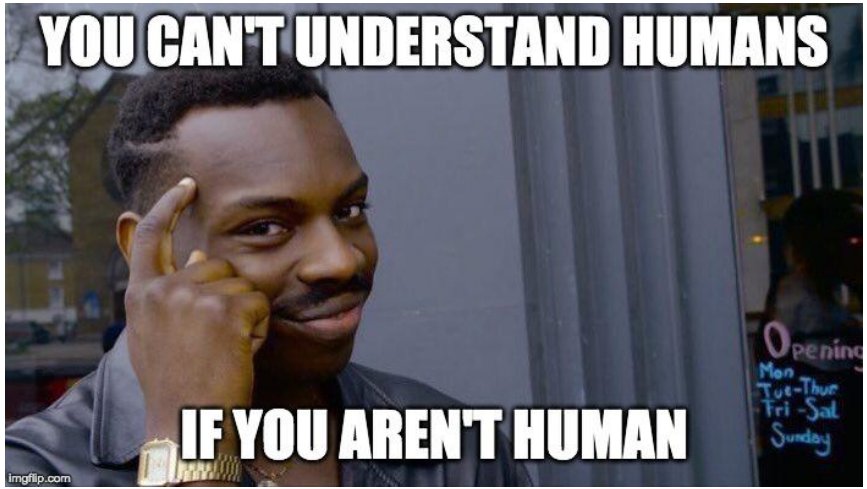
Testing failure on the login page
1. Open the login page
2. Fill in login form with incorrect credentials
3. It should not redirect to the homepage

Login form subroutine:
1. Fill in a username in the username field
2. Fill in a password in the password field
3. Click the login button

S

Create Feedback Loop with QA

- Verify changes with QA
- Establish communication between Dev and QA
  - Help QA understand:
    - How developers see the test cases
    - What can be tested
  - Helps developers understand:
    - How a human uses the product



S

Write the Test

Pseudocode:
Testing success on the login page:

1. Open the login page
2. Fill in login form with correct credentials
3. It should redirect to the homepage

Testing failure on the login page

1. Open the login page
2. Fill in login form with correct credentials
3. It should redirect to the homepage:

Login form subroutine:

1. Fill in a username in the username field
2. Fill in a password in the password field
3. Click the login button

```
def fill_login_form(self, username: str, password: str)
    login_page = self.login_page

    login_page.username_field.text = username
    login_page.password_field.test = password
    login_page.login_button.click()

def test_successful_login(self):
    # 1. Open the login page
    self.go_to_login_page()

    # 2. Fill in login form
    self.fill_login_form(username='user', password='pass')

    # 3. Expect to be on the homepage
    self.assertEquals('www.test-environment.com/homepage',
                      self.get_url(),
                      'Login was not successful')

def test_failed_login(self):
    # 1. Open the login page
    self.go_to_login_page()

    # 2. Fill in login form
    self.fill_login_form(username='wrong', password='wrong')

    # 3. Expect to be on the homepage
    self.assertEquals('www.test-environment.com/login',
                      self.get_url(),
                      'Login was not unexpectedly successful')
```

S

Challenge: Expecting an Application State

D

Can We Expect an Application State?

It Depends!

- How much time/infrastructure is required to spin up an instance of the app?
- How large is the test suite?
- How hard is it to generate all the required data for a given test case/suite?

Three recommended approaches...

D

Expected States - What to NEVER do

- Depend on manually created data
- Make tests dependent upon their execution order

Expected States - Pure Approach

- Spin up an entire test application environment for each test case.
- Each test runs in isolation against the same initial state of the application
- Pro:
  - Reliable: Perfectly identical test conditions for each run. No “bad data” failures
  - Test cases can run completely in parallel.
- Con:
  - Resources: Time + Hardware (even virtualized) = $
  - Load testing must be separately accounted for:
    - Does the application slow down/encounter errors as data accumulates?
    - Multiple parallel users of the application must be handled explicitly.

D

Expected States - Generate Everything

- Each test case generates ALL of the data it needs to complete.
- Pros:
  - Facilitates data accumulation/parallel usage load testing.
  - No data mutation issues.
  - Cases can be run completely in parallel
  - Single environment for all tests.
- Cons:
  - All data generation takes time. 🐢
  - Some “base” data may be hard to generate (such as a user login).
  - Inefficiency: Same data could be used across multiple test cases:
    - Ex: Processing multiple video conversions when the result of a single conversion could be used in multiple tests.

D

Expected States - Compromise (Reality)

- Inject “base” required data before starting tests.
- Use modules of closely related test cases.
- Pros:
  - Efficient data creation. Data can be generated for the module instead of each individual case.
  - Data/parallel usage load testing is facilitated.
  - Single environment for all tests.
- Cons:
  - Implementation is more complicated. (Logic needed to create/maintain data state in runs.)
  - Data-state errors are possible.
  - Parallelism can only occur at the module level.

D

Challenge: Making Tests Reliable and Reusable

J

Let's Test a Login Page

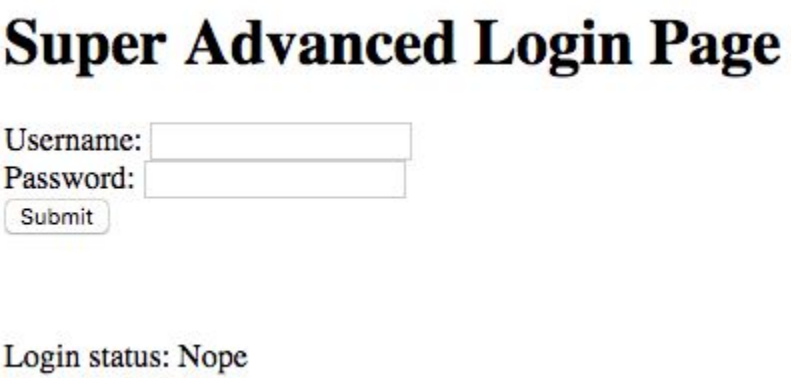
- We are all employees now at TestCo
- We have been tasked with testing our company's new login page
- Let's see how we can best structure our test's code

TestCo

Problems Today, Solutions Eventually

J

Our Login Page




```
<h1>Super Advanced Login Page</h1>
<form>
    <label>
        Username: <input id="username-input"
type="text">
    </label>
    <br>

    <label>
        Password: <input id="password-input"
type="password">
    </label>
    <br>
</form>
<label>
    <button
        id="submit-button"
        onClick="userValidate()">Submit</button>
</label>
...
<label>
    Login status: <span id="test-status">Nope</span>
</label>
```

J

Possible Solution

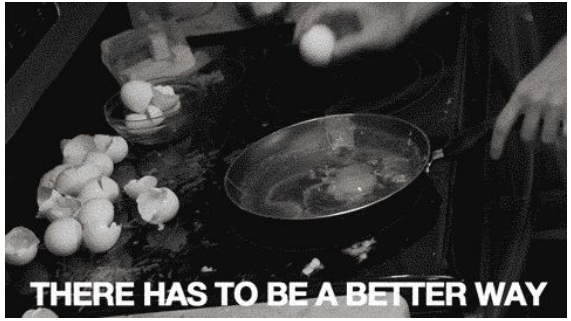
```
class TestLoginPage(unittest.TestCase):

    def test_successful_login(self):
        driver = webdriver.Chrome()
        driver.get("http://localhost:8000/example.html")

        username = driver.find_element_by_id('username-input')
        username.send_keys('admin')

        password = driver.find_element_by_id('password-input')
        password.send_keys('password')

        submit = driver.find_element_by_id('submit-button')
        submit.click()

        login_status = driver.find_element_by_id('test-status')
        self.assertEqual(login_status.text, 'Authenticated')
```

But Then Duplication Happens

```
class TestLoginPage(unittest.TestCase):

    def test_successful_login(self):
      …

   def test_unsuccessful_login(self):
        driver = webdriver.Chrome()
        driver.get("http://localhost:8000/example.html")

        username = driver.find_element_by_id('username-input')
        username.send_keys('incorrect-username')

        password = driver.find_element_by_id('password-input')
        password.send_keys('incorrect-password')

        submit = driver.find_element_by_id('submit-button')
        submit.click()

        login_status = driver.find_element_by_id('test-status')
        self.assertEqual(login_status.text, 'Failed')
```

And the Problems Don't Stop There!

- Our tests are not DRY
- What happens if the login's dom changes?
  - We'll have to update all of our selectors throughout all of our tests
- What happens if the login process changes?
  - Additional logic would have to be added to all future and existing tests
- What happens if we create new tests that have to login?



Design Patterns to the Rescue

- The **Page Object Model** (p.o.m) pattern is one where objects are used to represent pieces of the page under test
  - The reusable page pieces ensure that selector logic is funneled through one code path
- P.o.m can be augmented through use of something we call **Page Properties**
  - **Properties** are reusable objects that describe how to interact with common controls on a page
  - These **properties** help consolidate interaction logic into a singular code path

Basic Page Property Framework

```
class PageProperty:

    def __init__(self, driver: WebDriver, element:
WebElement):
        self.driver = driver
        self._element = element

    @property
    def element(self) -> WebElement:
        return self._element

    @property
    def text(self) -> str:
        return self.element.text

    def click(self) -> None:
        self.element.click()
```

```
class TextInputProperty(PageProperty):

    @PageProperty.text.setter
    def text(self, text: str):
        self.clear()
        self.element.send_keys(text)

    def clear(self):
        self.element.clear()


class LoginStatusProperty(PageProperty):

    @property
    def is_logged_in(self):
        return self.element.text == 'Authenticated'

    @property
    def login_failed(self):
        return self.element.text == 'Failed'
```

P.O.M for our Login Page

```
class LoginPage(Page):

    @property
    def username(self) -> TextInputProperty:
        return TextInputProperty(self.driver, self.driver.find_element_by_id('username-input'))

    @property
    def password(self) -> TextInputProperty:
        return TextInputProperty(self.driver, self.driver.find_element_by_id('password-input'))

    @property
    def submit_button(self) -> PageProperty:
        return PageProperty(self.driver, self.driver.find_element_by_id('submit-button'))

    @property
    def login_status(self) -> LoginStatusProperty:
        return LoginStatusProperty(self.driver, self.driver.find_element_by_id('test-status'))
```

Login With P.O.M

```
class TestLoginPage(unittest.TestCase):

    @property
    def login_page(self) -> LoginPage:
        return LoginPage(self.driver)

    def test_successful_login(self):
        self.driver = webdriver.Chrome()
        self.driver.get("http://localhost:8000/example.html")
        login_page = self.login_page

        login_page.username.text = 'admin'
        login_page.password.text = 'password'
        login_page.submit_button.click()

        self.assertTrue(login_page.login_status.is_logged_in)

    def test_unsuccessful_login(self):
        self.driver = webdriver.Chrome()
        self.driver.get("http://localhost:8000/example.html")
        login_page = self.login_page

        login_page.username.text = 'invalid-username'
        login_page.password.text = 'invalid-password'
        login_page.submit_button.click()

        self.assertTrue(login_page.login_status.login_failed)
```

Even Better

```
class TestLoginPage(unittest.TestCase):

    def setUp(self) -> None:
        super().setUp()
        self.driver = webdriver.Chrome()
        self.driver.get("http://localhost:8000/example.html")

    @property
    def login_page(self) -> LoginPage:
        return LoginPage(self.driver)

    def login(self, username: str, password: str):
        login_page = self.login_page

        login_page.username.text = username
        login_page.password.text = password
        login_page.submit_button.click()

    def test_successful_login(self):
        login_page = self.login_page

        self.login('admin', 'password')
        self.assertTrue(login_page.login_status.is_logged_in)

    def test_unsuccessful_login(self):
        login_page = self.login_page

        self.login('invalid-username', 'invalid-password')
        self.assertTrue(login_page.login_status.login_failed, 'Failed')
```

Wait, What Was That?

- https://github.com/hamologist/selenium-design-patterns
- https://github.com/hamologist/selenium-design-patterns-playground

Challenge-TIME!!!!!!

Development Time

- We want developers to spend more time writing the feature than testing it
- Document what common tasks are available
- UNIT TESTS!
- Test the “Critical Path”

S

Run Time

- Web Automation tests can be slow (Yes, we know)
- Write for parallelism
    - Each test should be independent
- Be smart with conditional waiting
    - Don’t “sleep” expecting for something in the UI to happen
- UNIT TESTS!

S

Code PaLOUsa 2019 Sponsors

Thank You!

Can I Get Those Links One More Time?

- https://github.com/hamologist/selenium-design-patterns
- https://github.com/hamologist/selenium-design-patterns-playground