TestZeus: Automate the lightning and thunder of Salesforce Testing

Robin Gupta

Robin.gupta@provartesting.com

Abstract

Software test automation for a SAAS (Software As a Service) is inherently complicated at various levels. While the development team does functional changes, the platform also evolves with time and poses exponential challenges for software testing teams with not only building out automation scripts, but also maintaining them over a period of time. Salesforce is a CRM (Client Relationship Management) SAAS offering and is a classic example of posing the aforementioned challenges to software development testing teams. Salesforce releases change the HTML, JavaScript, and CSS that make up the user interface. These changes break traditional automated tests and lead the testing teams down the rework spiral. This paper demonstrates the implementation of TestZeus an open-source framework, which uses the publicly available APIs to bridge the gap between test automation scripts and Salesforce platform, by reducing the reliance on the standard locator mechanisms.

Biography

Robin is a multi-talented engineering manager with close to 15 years of metrics driven approach to challenging assignments. He is successful at managing all aspects of delivery vis a vis Engineering and QA. Robin has a polymorphic leadership style with fluency in BFSI, EdTech, Retail, Healthcare and E-commerce domains. He has hands on experience working with startups and enterprises across geographies and time zones. Besides work he is a mentor at ADPList and Plato, and contributes to opensource via Selenium and TestZeus.

1 Introduction

Software testing has been an effective approach of ensuring the quality of web applications. In practice, software testers construct manual and automation scripts for testing out the web application. These automation scripts are created using open-source tools like Selenium, Playwright or Puppeteer etc. and work on the UI (User Interface) interaction layer to simulate end user behavior. These UI (User Interface) test automation tools identify the web elements (dropdowns, links, input boxes etc.) on the application under test, using locator strategies like identifying the xpath (XML Path Language), CSS (Cascading Style Sheets) or Ids. However, even with the advent of these tools, the test automation for SAAS (Software as A Service) applications like Salesforce remains a bane for the software testers due to the ever-changing nature of front-end code, as per platform level upgrades. Quarterly and annual service upgrades cause these web element locators to break and the testing team to focus on maintenance for the automation scripts. Additionally, waiting for the page to load in a multi-tier web architecture can add extra layers of failure for a test automation framework. Therefore, in a large test automation base, the maintenance of these frameworks can exceed the release window of the application itself, rendering the coverage useless.

In order to solve this trident problem of accelerating automation development, stabilizing the locator strategy and reducing the maintenance effort, we explored the dynamic creation of locators via the TestZeus framework using Salesforce's publicly available UI API and observed at least 30% gains in execution times, along with 60% reduction in test creation and maintenance effort.

2 Salesforce UI API

Salesforce UI API provides data and metadata around layouts, field-level security, fields and types in a well formatted JSON response. This is the same API which is consumed by the Salesforce front end system (Lightning Design System) to render the UI at run time. Along with the layout and design information about the front end, UI API also provides record level information. For example, if we query the UI API for an ACCOUNT object (Example: RecordId as 001R0000003GeJ1IAK):

GET /ui-api/record-ui/001R0000003GeJ1IAK

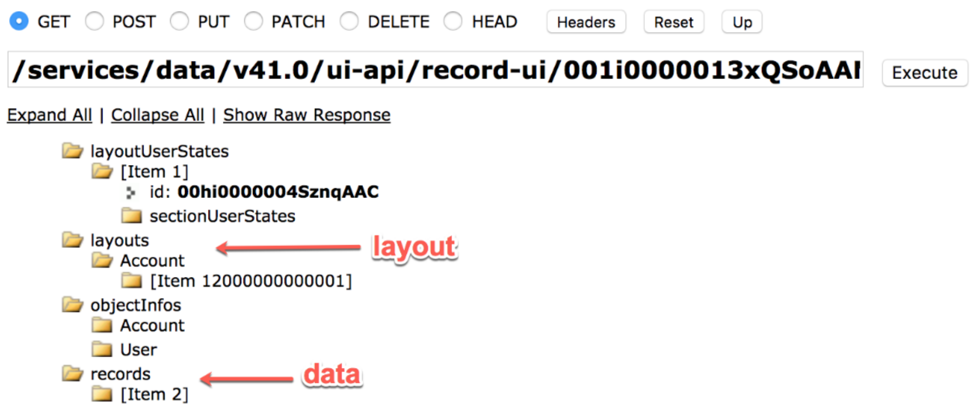
We will get the below crucial information in a neat JSON response:

1. Fields on the ACCOUNT object like First Name, Last Name, Phone, Address etc.
2. Placement and layout related information of the aforementioned fields such as which field lies on which section of the page layout.
3. Field types of the various fields on the page
4. Exact values of the fields for the record id passed in the API call. For example: First Name as "Robin", Last Name as "Gupta", Address as "Bangalore" etc.

There are many use cases where customers build custom mobile or web apps that use Salesforce data and metadata—sometimes they want to just replicate all or partial Salesforce UI inside those apps. But Salesforce UIs are complex. The correct UI for any given user depends on various rules, permissions, and page layout configurations. All of these dependencies can be easily changed by an admin (or anyone with appropriate permissions) at any time to better fit business needs. It used to be very hard to build dynamic UIs and display them in custom apps that take care of all the rules and updates whenever changes are made.

That's where the UI API comes in. Its main purpose is to make it simple to replicate all or part of a Salesforce UI according to rules and permissions inside a given custom app. Since UIs also allow the users to edit, update, or delete the data, the UI API further provides endpoints to even perform CRUD operations corresponding to each layout with ease. This data is intended for developers to build a dynamic UI for native and mobile apps. It can also be consumed for designing and implementing custom applications which interact with Salesforce databases.

Examples of the response from UI API are listed below in Diagram A and Diagram B.

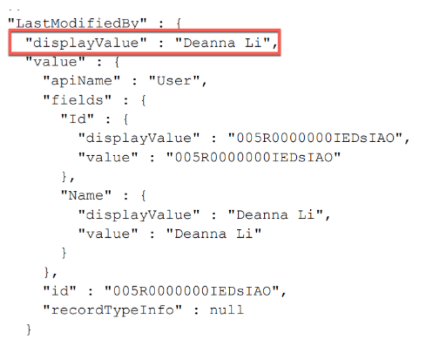


```
/services/data/v41.0/ui-api/record-ui/001i0000013xQSoAAI

layoutUserStates
  [Item 1]
    id: 00hi0000004SznqAAC
    sectionUserStates
layouts
  Account
    [Item 12000000000001]
objectInfos
  Account
  User
records
  [Item 2]
```

(Diagram A).

The above diagram depicts the high-level structure of the response received from UI API for a sample record. Notably, it contains the layout information and the data points for the record.



```
..
"LastModifiedBy" : {
  "displayValue" : "Deanna Li",
  "value" : {
    "apiName" : "User",
    "fields" : {
      "Id" : {
        "displayValue" : "005R0000000IEDsIAO",
        "value" : "005R0000000IEDsIAO"
      },
      "Name" : {
        "displayValue" : "Deanna Li",
        "value" : "Deanna Li"
      }
    },
    "id" : "005R0000000IEDsIAO",
    "recordTypeInfo" : null
  }
```

(Diagram B).

The above diagram shows the exact displayed values for the record as received from a sample UI API response.

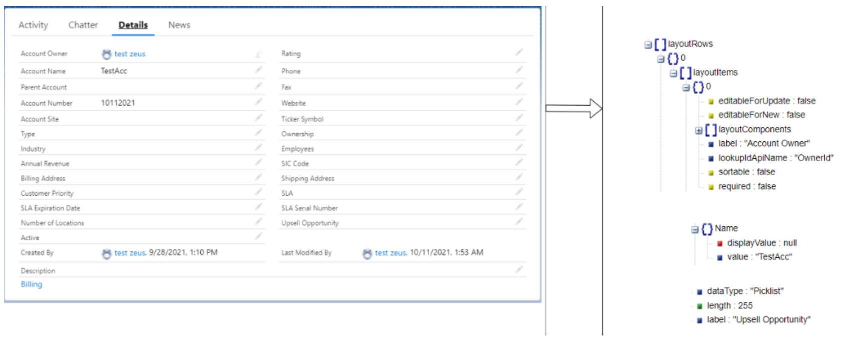
3 TestZeus

TestZeus is an open-source test automation framework built specifically for the Salesforce platform.

Based on the information from Salesforce's publicly available UI API, TestZeus creates the locators automatically at run time to exponentially cut down the build and maintenance times for test automation scripts. In addition to the bespoke locator strategy, TestZeus also supports waiting mechanisms built specifically for Salesforce and custom components to handle API testing, Email integrations and Page object implementation.

3.1 Auto-Locators

The core of TestZeus works on automatically creating web element locators based on the response received from UI API for a certain object or record. TestZeus parses the UI API and processes the JSON response for labels, datatypes and sections to create the contextual actions and locators for the UI elements on the fly as depicted in Diagram C below:



Account Owner	test zeus	Rating	
Account Name	TestAcc	Phone	
Parent Account		Fax	
Account Number	10112021	Website	
Account Site		Ticker Symbol	
Type		Ownership	
Industry		Employees	
Annual Revenue		SIC Code	
Billing Address		Shipping Address	
Customer Priority		SLA	
SLA Expiration Date		SLA Serial Number	
Number of Locations		Upsell Opportunity	
Active			
Created By	test zeus, 9/28/2021, 1:10 PM	Last Modified By	test zeus, 10/11/2021, 1:53 AM
Description			
Billing			

```
[ ] layoutRows
  { } 0
    [ ] layoutItems
      { } 0
        editableForUpdate : false
        editableForNew : false
        [ ] layoutComponents
        label : "Account Owner"
        lookupIdApiName : "OwnerId"
        sortable : false
        required : false

      { } Name
        displayValue : null
        value : "TestAcc"

      dataType : "Picklist"
      length : 255
      label : "Upsell Opportunity"
```

(Diagram C).

TestZeus scrapes the sections, labels and datatypes from the UI API to create the stable locators and based on the datatypes, contextually provides actions to interact with the UI.

Here is the flow of events in programmatic fashion:

1. Hit the UI API via a connected app
2. Get the list of sections for the object
3. Recursively get the field types and labels for all the sections on the page
4. Create a data structure for the user interactions based on the test flow

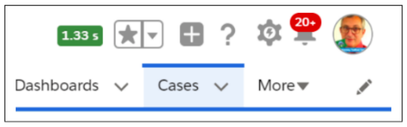
As the locators are created at run time based on the above data, user need not create page objects explicitly for the test case.

These locators are passed onto standard Selenium based web driver to accurately perform the user interaction based on the test flow.

Also, the UI API can be utilized for CRUD (Create, Retrieve, Update and Delete) operations, therefore testers can not only view and assert the data, they can modify the field values using TestZeus as well.

3.2 Experienced Page Time

Experienced Page Time (EPT) is a performance metric Salesforce uses in Lightning to measure page load time. EPT measures how long it takes for a page to load into a state that a user can meaningfully interact with. The EPT is measured as the time from the page start to when no more activity occurs for at least two frames (~33 milliseconds). The two extra frames help to avoid false positives due to asynchronous calls. These calls include any XHR activity, any storage activity, or any user interaction or client-side work of any kind in the main JavaScript thread. Example EPT of 1.33 seconds is shown in the Diagram D below:



(Diagram D).

3.3 Intelligent Wait Mechanism

In addition to utilizing the UI API for dynamic creation of locators, TestZeus also utilizes the EPT (Experience Page Time) as described above to intelligently wait for the web page to load. Basically, TestZeus continuously polls the EPT value for 20 seconds and waits for it to be a value greater than zero, meaning that the web page has loaded completely, and is interact-able as published by Salesforce's front-end system. This has been implemented using the value retrieved from console as part of the browser interaction.

3.4 Example Usage

TestZeus can be utilized as the boiler plate framework for starting test automation or can be imported as a MAVEN dependency in an existing test automation framework. Here's a sample test case built using TestZeus, which logs into a Salesforce instance, and creates a new account without having to write explicit page objects or web element locators:

```
//Create a new instance of the SFPageBase class

SFPageBase pb = new SFPageBase(driver);

// Use methods from TestZeus as below for Navigation to login page

pb.openHomepage("https://testzeus2-dev-ed.my.salesforce.com");

pb.maximize();

// Or Use the webdriver implementations: Example for Submitting user id,

// password and logging in

driver.findElement(By.id("username")).sendKeys("UIusername@gmail.com");

driver.findElement(By.id("password")).sendKeys("UIpassword");

WebElement loginbutton = driver.findElement(By.id("Login"));

pb.safeClick(loginbutton);

pb.appLauncher("Account");

WebElement newbutton = driver.findElement(By.xpath("//a[@title='New']"));

pb.safeClick(newbutton);

// We fetch all the labels and datatype from UI API here for a certain record

String recordid = "0015g00000S9lfUAAR";

pb.uiApiParser(recordid);
```

```
// Form data can be passed directly on the new sObject creation screen

        pb.formValueFiller("Account Name", "AccountCreatedOn : " + pb.getCurrentDateTimeStamp());

        WebElement savebutton = driver.findElement(By.xpath("//button[@name='SaveEdit']"));

        pb.safeClick(savebutton);
```

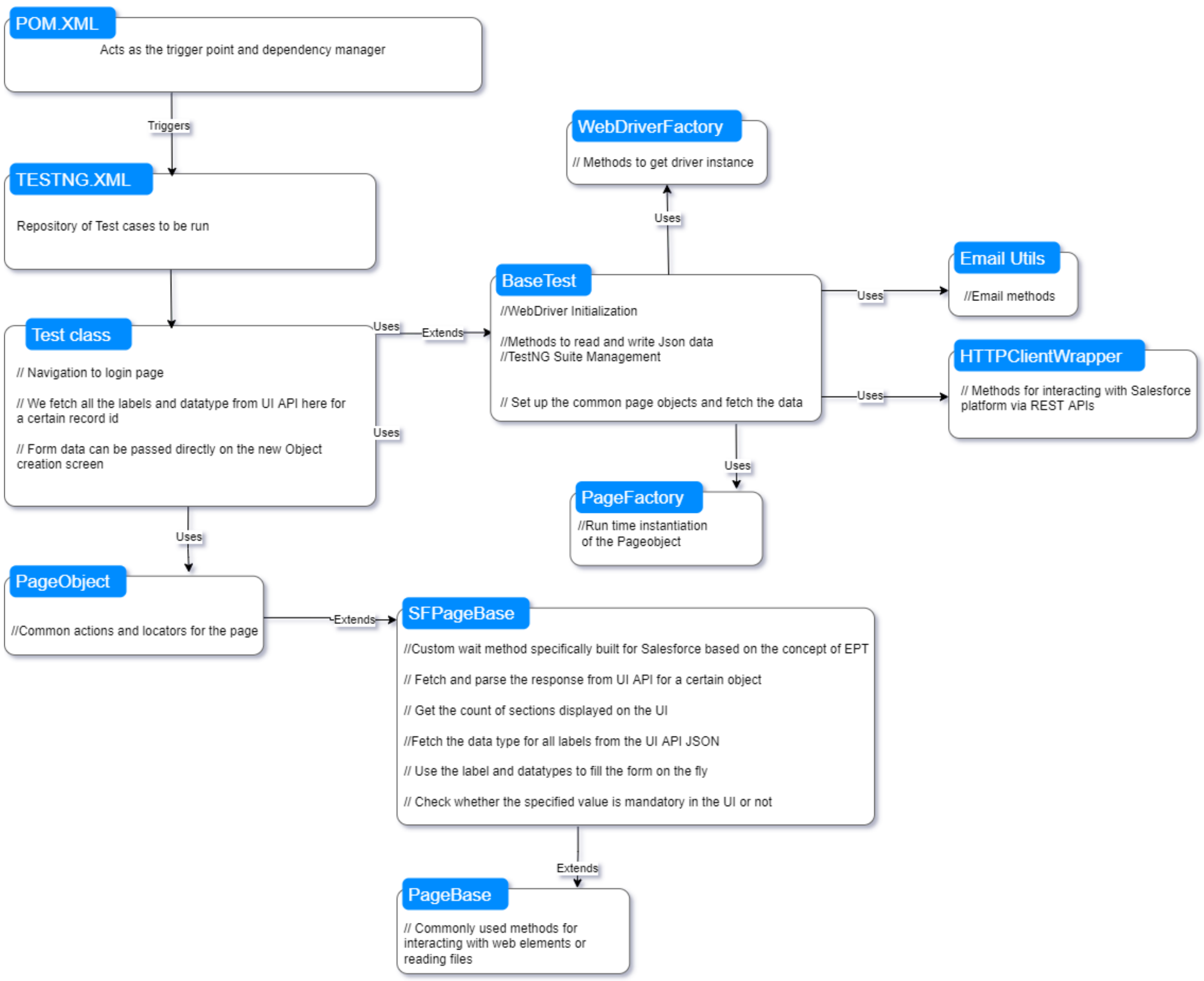
3.5 Results

TestZeus brings value and results at three levels as below:

1. Accelerates the Test automation effort by removing the need for writing explicit locators
2. Adding stability via automatically waiting for the page to load
3. Reducing the maintenance effort for feature and platform level changes.

3.6 Architecture

Diagram E below shows the high-level architecture of the TestZeus framework:


POM.XML
Acts as the trigger point and dependency manager
Triggers
TESTNG.XML
Repository of Test cases to be run
Test class
// Navigation to login page
// We fetch all the labels and datatype from UI API here for a certain record id
// Form data can be passed directly on the new Object creation screen
Uses
Extends
Uses
Uses
PageObject
//Common actions and locators for the page
Extends
WebDriverFactory
// Methods to get driver instance
Uses
BaseTest
//WebDriver Initialization
//Methods to read and write Json data
//TestNG Suite Management
// Set up the common page objects and fetch the data
Uses
Email Utils
//Email methods
Uses
HTTPClientWrapper
// Methods for interacting with Salesforce platform via REST APIs
Uses
PageFactory
//Run time instantiation of the Pageobject
SFPageBase
//Custom wait method specifically built for Salesforce based on the concept of EPT
// Fetch and parse the response from UI API for a certain object
// Get the count of sections displayed on the UI
//Fetch the data type for all labels from the UI API JSON
// Use the label and datatypes to fill the form on the fly
// Check whether the specified value is mandatory in the UI or not
Extends
PageBase
// Commonly used methods for interacting with web elements or reading files

4 Challenges and Limitations

As with any technological advancement, TestZeus is also limited by its coverage and Salesforce's publicly available APIs. Specifically, auto-locator strategy was briefly touched in the Section 3.1 above, it is nowhere near the point of hundred percent test coverage for the whole Salesforce ecosystem encompassing the Lightning Design System, Lightning Web Components, Standard and Custom objects, Installed packages and Visualforce pages. A paradigm shift in application testing focused on more granular validations combined with more simplified test case creation may result in a convergent satisfaction of this limitation.

5 Conclusion

In this paper we present a novel approach to test automation for Salesforce ecosystem – called TestZeus. TestZeus uses an API first approach to generate web element locators, which are dynamic in nature and change automatically based on the page layout changes. Internally, TestZeus also intelligently waits for web pages to load and consumes Selenium Webdriver APIs to drive the browser and mimic user behavior. There were numerous other observations and feedback received from the technical implementation. We believe there is scope to considerably improve the coverage of the Salesforce ecosystem by modelling the out of the box objects and features offered by Salesforce. The concept and the implementation of TestZeus were very well received by the software testing and open-source community. Current efforts are in deploying TestZeus in other scenarios within multiple projects and improve the tool's capability particularly around various kind of web systems built using Salesforce.

The same approach of using metadata level APIs can be utilized in other SAAS applications like Oracle, Workday, JIRA and ServiceNow as well.

References

Selenium Webdriver, http://www.seleniumhq.org/projects/webdriver/ (Online, retrieved 15 June 2022).

Salesforce UI API, https://developer.salesforce.com/docs/atlas.en-us.uiapi.meta/uiapi/ui_api_get_started.htm (Online, retrieved 15 June 2022).

Salesforce EPT, https://help.salesforce.com/s/articleView?id=sf.technical_requirements_ept.htm&type=5 (Online, retrieved 15 June 2022).

TestZeus, http://testzeus.com/ (Online, retrieved 15 June 2022).

Apache Maven, https://maven.apache.org/ (Online, retrieved 15 June 2022).