**OrangeHRM Automation Framework Documentation**

**Table of Contents**

1. [Overview](https://claude.ai/chat/991abd59-dd1c-4c91-bc61-7dd928460a68#overview)
2. [Data Model Creation](https://claude.ai/chat/991abd59-dd1c-4c91-bc61-7dd928460a68#data-model-creation)
3. [Page Object Implementation](https://claude.ai/chat/991abd59-dd1c-4c91-bc61-7dd928460a68#page-object-implementation)
4. [GetPageData Function](https://claude.ai/chat/991abd59-dd1c-4c91-bc61-7dd928460a68#getpagedata-function)
5. [Usage in Tests](https://claude.ai/chat/991abd59-dd1c-4c91-bc61-7dd928460a68#usage-in-tests)
6. [Best Practices](https://claude.ai/chat/991abd59-dd1c-4c91-bc61-7dd928460a68#best-practices)
7. [Troubleshooting](https://claude.ai/chat/991abd59-dd1c-4c91-bc61-7dd928460a68#troubleshooting)

**1. Overview**

**Framework Architecture**

The OrangeHRM automation framework implements a robust Page Object Model (POM) pattern with integrated data generation and validation capabilities. This framework automates the employee management workflows on the OrangeHRM demo site using:

* **Selenium WebDriver**: For browser automation and element interactions
* **Bogus Library**: For generating realistic test data dynamically
* **Page Object Model**: For maintaining clean separation between test logic and page interactions
* **Data Retrieval Pattern**: Custom GetPageData functions for extracting entered data from forms

**Key Components**

Framework Structure:

├── Models/

│ ├── EmployeeData.cs

│ └── PersonalDetails.cs

├── PageObjects/

│ ├── LoginPage.cs

│ ├── AddEmployeePage.cs

│ └── EmployeeDetailsPage.cs

├── DataGenerators/

│ └── EmployeeDataGenerator.cs

├── Tests/

│ └── EmployeeManagementTests.cs

└── Utilities/

└── AssertAndLog.cs

**Data Flow**

1. **Generation**: Bogus creates random, realistic data
2. **Entry**: Page objects fill forms with generated data
3. **Retrieval**: GetPageData functions extract entered values
4. **Validation**: Test classes compare generated vs. retrieved data

**2. Data Model Creation**

**Employee Data Model**

Create strongly-typed models to represent the data structure for employee information:

public class EmployeeData

{

public string FirstName { get; set; }

public string MiddleName { get; set; }

public string LastName { get; set; }

public string EmployeeId { get; set; }

public string Username { get; set; }

public string Password { get; set; }

public PersonalDetails PersonalDetails { get; set; }

}

public class PersonalDetails

{

public string Nickname { get; set; }

public string OtherId { get; set; }

public string DriversLicenseNumber { get; set; }

public DateTime LicenseExpiryDate { get; set; }

public string SSNNumber { get; set; }

public string SINNumber { get; set; }

public string Nationality { get; set; }

public string MaritalStatus { get; set; }

public DateTime DateOfBirth { get; set; }

public string Gender { get; set; }

}

**Data Generator Implementation**

Utilize Bogus to create realistic test data:

public class EmployeeDataGenerator

{

private readonly Faker<EmployeeData> \_faker;

public EmployeeDataGenerator()

{

\_faker = new Faker<EmployeeData>()

.RuleFor(e => e.FirstName, f => f.Name.FirstName())

.RuleFor(e => e.MiddleName, f => f.Name.FirstName())

.RuleFor(e => e.LastName, f => f.Name.LastName())

.RuleFor(e => e.EmployeeId, f => f.Random.Number(10000, 99999).ToString())

.RuleFor(e => e.Username, (f, e) => $"{e.FirstName.ToLower()}.{e.LastName.ToLower()}")

.RuleFor(e => e.Password, f => f.Internet.Password(12, true))

.RuleFor(e => e.PersonalDetails, f => GeneratePersonalDetails(f));

}

public EmployeeData Generate() => \_faker.Generate();

}

**3. Page Object Implementation**

**Base Page Object Structure**

Each page object should inherit from a base class and implement data entry and retrieval methods:

public class AddEmployeePage : BasePage

{

private readonly IWebDriver \_driver;

// Element Locators

private By FirstNameInput => By.Name("firstName");

private By MiddleNameInput => By.Name("middleName");

private By LastNameInput => By.Name("lastName");

private By EmployeeIdInput => By.XPath("//label[text()='Employee Id']/../..//input");

private By SaveButton => By.XPath("//button[@type='submit']");

public AddEmployeePage(IWebDriver driver) : base(driver)

{

\_driver = driver;

}

// Data Entry Method

public void FillEmployeeForm(EmployeeData data)

{

WaitAndSendKeys(FirstNameInput, data.FirstName);

WaitAndSendKeys(MiddleNameInput, data.MiddleName);

WaitAndSendKeys(LastNameInput, data.LastName);

ClearAndSendKeys(EmployeeIdInput, data.EmployeeId);

Click(SaveButton);

}

// Data Retrieval Method

public EmployeeData GetPageData()

{

// Implementation detailed in next section

}

}

**Helper Methods for Page Objects**

public abstract class BasePage

{

protected IWebDriver Driver;

protected WebDriverWait Wait;

protected BasePage(IWebDriver driver)

{

Driver = driver;

Wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));

}

protected void WaitAndSendKeys(By locator, string text)

{

var element = Wait.Until(ExpectedConditions.ElementIsVisible(locator));

element.Clear();

element.SendKeys(text);

}

protected string GetValue(By locator)

{

var element = Wait.Until(ExpectedConditions.ElementIsVisible(locator));

return element.GetAttribute("value") ?? element.Text;

}

}

**4. GetPageData Function**

**Implementation Pattern**

The GetPageData function is crucial for retrieving entered data from the form for validation purposes:

public class EmployeeDetailsPage : BasePage

{

// Locators for reading data

private By FirstNameField => By.Name("firstName");

private By MiddleNameField => By.Name("middleName");

private By LastNameField => By.Name("lastName");

private By EmployeeIdField => By.XPath("//label[text()='Employee Id']/../..//input");

private By NicknameField => By.XPath("//label[text()='Nickname']/../..//input");

public EmployeeData GetPageData()

{

var employeeData = new EmployeeData

{

FirstName = GetFieldValue(FirstNameField),

MiddleName = GetFieldValue(MiddleNameField),

LastName = GetFieldValue(LastNameField),

EmployeeId = GetFieldValue(EmployeeIdField),

PersonalDetails = GetPersonalDetails()

};

return employeeData;

}

private PersonalDetails GetPersonalDetails()

{

return new PersonalDetails

{

Nickname = GetFieldValue(NicknameField),

OtherId = GetFieldValue(By.XPath("//label[text()='Other Id']/../..//input")),

DriversLicenseNumber = GetFieldValue(By.XPath("//label[contains(text(),'License Number')]/../..//input")),

SSNNumber = GetFieldValue(By.XPath("//label[text()='SSN Number']/../..//input")),

Nationality = GetDropdownSelectedValue(By.XPath("//label[text()='Nationality']/../..//select")),

MaritalStatus = GetDropdownSelectedValue(By.XPath("//label[text()='Marital Status']/../..//select")),

Gender = GetRadioButtonValue("gender"),

DateOfBirth = ParseDateField(By.XPath("//label[text()='Date of Birth']/../..//input"))

};

}

private string GetFieldValue(By locator)

{

try

{

var element = Wait.Until(ExpectedConditions.ElementIsVisible(locator));

return element.GetAttribute("value") ?? element.Text ?? string.Empty;

}

catch (WebDriverTimeoutException)

{

return string.Empty;

}

}

private string GetDropdownSelectedValue(By locator)

{

var selectElement = new SelectElement(Driver.FindElement(locator));

return selectElement.SelectedOption.Text;

}

private DateTime ParseDateField(By locator)

{

var dateString = GetFieldValue(locator);

return DateTime.TryParse(dateString, out var date) ? date : DateTime.MinValue;

}

}

**Advanced Retrieval Patterns**

**Handling Dynamic Elements**

public string GetDynamicFieldValue(string fieldLabel)

{

var xpath = $"//label[contains(text(),'{fieldLabel}')]/following-sibling::div//input";

return GetFieldValue(By.XPath(xpath));

}

**Retrieving Table Data**

public List<Dictionary<string, string>> GetTableData(By tableLocator)

{

var table = Driver.FindElement(tableLocator);

var headers = table.FindElements(By.TagName("th"))

.Select(h => h.Text).ToList();

var rows = table.FindElements(By.XPath(".//tbody/tr"));

var tableData = new List<Dictionary<string, string>>();

foreach (var row in rows)

{

var cells = row.FindElements(By.TagName("td"));

var rowData = new Dictionary<string, string>();

for (int i = 0; i < headers.Count && i < cells.Count; i++)

{

rowData[headers[i]] = cells[i].Text;

}

tableData.Add(rowData);

}

return tableData;

}

**5. Usage in Tests**

**Test Class Implementation**

[TestClass]

public class EmployeeManagementTests : BaseTest

{

private EmployeeDataGenerator \_dataGenerator;

private LoginPage \_loginPage;

private AddEmployeePage \_addEmployeePage;

private EmployeeDetailsPage \_employeeDetailsPage;

[TestInitialize]

public void TestSetup()

{

\_dataGenerator = new EmployeeDataGenerator();

\_loginPage = new LoginPage(Driver);

\_addEmployeePage = new AddEmployeePage(Driver);

\_employeeDetailsPage = new EmployeeDetailsPage(Driver);

// Login to application

\_loginPage.Login("Admin", "admin123");

}

[TestMethod]

public void ValidateEmployeeCreation()

{

// Step 1: Generate test data

var generatedData = \_dataGenerator.Generate();

// Step 2: Navigate and fill form

NavigateToAddEmployee();

\_addEmployeePage.FillEmployeeForm(generatedData);

// Step 3: Wait for save and retrieve entered data

WaitForPageLoad();

var retrievedData = \_employeeDetailsPage.GetPageData();

// Step 4: Validate using Assert.That

Assert.That(retrievedData.FirstName, Is.EqualTo(generatedData.FirstName),

"First name mismatch");

Assert.That(retrievedData.LastName, Is.EqualTo(generatedData.LastName),

"Last name mismatch");

Assert.That(retrievedData.EmployeeId, Is.EqualTo(generatedData.EmployeeId),

"Employee ID mismatch");

}

[TestMethod]

public void ValidateEmployeeDetailsWithLogging()

{

var generatedData = \_dataGenerator.Generate();

NavigateToAddEmployee();

\_addEmployeePage.FillEmployeeForm(generatedData);

WaitForPageLoad();

var retrievedData = \_employeeDetailsPage.GetPageData();

// Using AssertAndLog for detailed validation

AssertAndLog.AreEqual(generatedData.FirstName, retrievedData.FirstName,

"Validating First Name");

AssertAndLog.AreEqual(generatedData.MiddleName, retrievedData.MiddleName,

"Validating Middle Name");

AssertAndLog.AreEqual(generatedData.LastName, retrievedData.LastName,

"Validating Last Name");

// Validate nested objects

AssertAndLog.IsNotNull(retrievedData.PersonalDetails,

"Personal Details should be populated");

if (generatedData.PersonalDetails != null)

{

AssertAndLog.AreEqual(

generatedData.PersonalDetails.Nationality,

retrievedData.PersonalDetails.Nationality,

"Validating Nationality");

}

}

}

**AssertAndLog Utility Implementation**

public static class AssertAndLog

{

private static readonly ILogger Logger = LogManager.GetCurrentClassLogger();

public static void AreEqual<T>(T expected, T actual, string message)

{

try

{

Assert.That(actual, Is.EqualTo(expected), message);

Logger.Info($"✓ Assertion Passed: {message} | Expected: {expected} | Actual: {actual}");

}

catch (AssertionException ex)

{

Logger.Error($"✗ Assertion Failed: {message} | Expected: {expected} | Actual: {actual}");

throw;

}

}

public static void IsNotNull(object value, string message)

{

try

{

Assert.That(value, Is.Not.Null, message);

Logger.Info($"✓ Not Null Assertion Passed: {message}");

}

catch (AssertionException ex)

{

Logger.Error($"✗ Not Null Assertion Failed: {message}");

throw;

}

}

}

**6. Best Practices**

**Data Generation Best Practices**

1. **Consistent Data Patterns**
   * Use seed values for reproducible tests when debugging
   * Generate data that matches business rules and constraints
2. var faker = new Faker<EmployeeData>().UseSeed(12345);
3. **Data Boundary Testing**
   * Create specific generators for edge cases
4. public EmployeeData GenerateWithMaxLengthValues()
5. {
6. return new EmployeeData
7. {
8. FirstName = new string('A', 50), // Max length
9. LastName = new string('Z', 50)
10. };
11. }
12. **Data Pool Management**
    * Maintain generated data for cleanup operations
13. private readonly List<EmployeeData> \_createdEmployees = new List<EmployeeData>();
14. [TestCleanup]
15. public void Cleanup()
16. {
17. foreach(var employee in \_createdEmployees)
18. {
19. DeleteEmployee(employee.EmployeeId);
20. }
21. }

**Page Object Best Practices**

1. **Wait Strategies**
2. public void WaitForDataLoad()
3. {
4. Wait.Until(driver =>
5. !driver.FindElement(By.ClassName("loader")).Displayed);
6. }
7. **Robust Element Location**
8. private IWebElement FindElementSafely(By locator)
9. {
10. try
11. {
12. return Wait.Until(ExpectedConditions.ElementIsVisible(locator));
13. }
14. catch (WebDriverTimeoutException)
15. {
16. Logger.Warn($"Element not found: {locator}");
17. return null;
18. }
19. }
20. **Data Validation Patterns**
21. public bool CompareEmployeeData(EmployeeData expected, EmployeeData actual)
22. {
23. return expected.FirstName == actual.FirstName &&
24. expected.LastName == actual.LastName &&
25. expected.EmployeeId == actual.EmployeeId;
26. }

**GetPageData Function Best Practices**

1. **Null Safety**
2. public string GetSafeValue(By locator)
3. {
4. var element = FindElementSafely(locator);
5. return element?.GetAttribute("value") ?? string.Empty;
6. }
7. **Data Type Conversion**
8. public T GetFieldValueAs<T>(By locator)
9. {
10. var value = GetFieldValue(locator);
11. return (T)Convert.ChangeType(value, typeof(T));
12. }
13. **Caching Retrieved Data**
14. private EmployeeData \_cachedData;
15. public EmployeeData GetPageData(bool useCache = true)
16. {
17. if (useCache && \_cachedData != null)
18. return \_cachedData;
20. \_cachedData = RetrieveDataFromPage();
21. return \_cachedData;
22. }

**7. Troubleshooting**

**Common Issues and Solutions**

**Issue 1: Stale Element Reference**

**Problem**: Element becomes stale after page refresh or DOM update

// Solution: Re-find element before interaction

public string GetValueWithRetry(By locator, int maxRetries = 3)

{

for (int i = 0; i < maxRetries; i++)

{

try

{

return Driver.FindElement(locator).GetAttribute("value");

}

catch (StaleElementReferenceException)

{

Thread.Sleep(500);

}

}

throw new Exception($"Unable to get value after {maxRetries} retries");

}

**Issue 2: Data Mismatch Due to Formatting**

**Problem**: Generated data doesn't match retrieved data due to formatting

// Solution: Normalize data before comparison

public class DataNormalizer

{

public static string NormalizePhoneNumber(string phone)

{

return Regex.Replace(phone, @"[^\d]", "");

}

public static DateTime NormalizeDate(DateTime date)

{

return date.Date; // Remove time component

}

public static string NormalizeName(string name)

{

return name?.Trim().ToUpperInvariant() ?? string.Empty;

}

}

**Issue 3: Timing Issues with GetPageData**

**Problem**: Data retrieval happens before form save completes

// Solution: Implement explicit waits

public void WaitForSaveComplete()

{

// Wait for success message

Wait.Until(ExpectedConditions.ElementIsVisible(

By.ClassName("oxd-toast-content")));

// Wait for URL change

Wait.Until(driver => driver.Url.Contains("/viewPersonalDetails"));

// Additional wait for data load

Thread.Sleep(1000); // Last resort if needed

}

**Issue 4: Dropdown Values Not Matching**

**Problem**: Dropdown text vs value mismatch

// Solution: Get both text and value

public DropdownData GetDropdownData(By locator)

{

var select = new SelectElement(Driver.FindElement(locator));

return new DropdownData

{

SelectedText = select.SelectedOption.Text,

SelectedValue = select.SelectedOption.GetAttribute("value"),

AllOptions = select.Options.Select(o => o.Text).ToList()

};

}

**Debugging Techniques**

1. **Screenshot on Failure**
2. [TestCleanup]
3. public void TakeScreenshotOnFailure()
4. {
5. if (TestContext.CurrentTestOutcome == TestOutcome.Failed)
6. {
7. var screenshot = ((ITakesScreenshot)Driver).GetScreenshot();
8. screenshot.SaveAsFile($"failure\_{TestContext.TestName}\_{DateTime.Now:yyyyMMdd\_HHmmss}.png");
9. }
10. }
11. **Data Comparison Logging**
12. public void LogDataComparison(EmployeeData expected, EmployeeData actual)
13. {
14. var differences = new List<string>();
16. if (expected.FirstName != actual.FirstName)
17. differences.Add($"FirstName: Expected '{expected.FirstName}' but got '{actual.FirstName}'");
19. if (expected.LastName != actual.LastName)
20. differences.Add($"LastName: Expected '{expected.LastName}' but got '{actual.LastName}'");
22. if (differences.Any())
23. {
24. Logger.Error("Data Comparison Failed:");
25. differences.ForEach(d => Logger.Error($" - {d}"));
26. }
27. }
28. **Element State Verification**
29. public void DebugElementState(By locator)
30. {
31. var element = Driver.FindElement(locator);
32. Logger.Debug($"Element State for {locator}:");
33. Logger.Debug($" - Displayed: {element.Displayed}");
34. Logger.Debug($" - Enabled: {element.Enabled}");
35. Logger.Debug($" - Selected: {element.Selected}");
36. Logger.Debug($" - Tag Name: {element.TagName}");
37. Logger.Debug($" - Value: {element.GetAttribute("value")}");
38. Logger.Debug($" - Text: {element.Text}");
39. }

**Performance Optimization**

1. **Batch Data Retrieval**
2. public Dictionary<string, string> GetAllFormData()
3. {
4. var formData = new Dictionary<string, string>();
5. var inputs = Driver.FindElements(By.TagName("input"));
7. foreach (var input in inputs)
8. {
9. var name = input.GetAttribute("name");
10. if (!string.IsNullOrEmpty(name))
11. {
12. formData[name] = input.GetAttribute("value");
13. }
14. }
16. return formData;
17. }
18. **Parallel Data Validation**
19. public void ValidateInParallel(EmployeeData expected, EmployeeData actual)
20. {
21. Parallel.Invoke(
22. () => Assert.That(actual.FirstName, Is.EqualTo(expected.FirstName)),
23. () => Assert.That(actual.LastName, Is.EqualTo(expected.LastName)),
24. () => Assert.That(actual.EmployeeId, Is.EqualTo(expected.EmployeeId))
25. );
26. }

**Conclusion**

This framework provides a robust solution for automating OrangeHRM with dynamic data generation and validation. The GetPageData pattern ensures reliable data retrieval for comprehensive test validation, while the Bogus integration provides realistic test data. Following these guidelines and best practices will result in maintainable, scalable, and reliable test automation.