**Form Data Retrieval Documentation**

**Table of Contents**

1. [**Overview**](#overview)
2. [**Data Model Creation**](#data-model-creation)
3. [**Page Object Implementation**](#page-object-implementation)
4. [**GetPageData Function**](#getpagedata-function)
5. [**Usage in Tests**](#usage-in-tests)
6. [**Best Practices**](#best-practices)
7. [**Troubleshooting**](#troubleshooting)

**Overview**

This documentation explains how to retrieve data from form pages using the Page Object Model pattern. The approach involves creating data models, implementing GetPageData() functions in page objects, and using these in tests for data verification and comparison.

Key Components

* Data Model: C# class that represents form data structure
* Page Object: Contains element locators and interaction methods
* GetPageData(): Function that retrieves current form values
* Test Implementation: Uses the retrieved data for assertions and verification

**Data Model Creation**

**1. Basic Data Model Structure**

**csharp**

**using System;**

**using System.Collections.Generic;**

**using System.ComponentModel.DataAnnotations;**

**namespace YourProject.Models**

**{**

**public class EmployeeFormData**

**{**

***// Personal Information***

**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**public string Email { get; set; }**

**public string PhoneNumber { get; set; }**

***// Employment Details***

**public string EmployeeId { get; set; }**

**public string Department { get; set; }**

**public string JobTitle { get; set; }**

**public string StartDate { get; set; }**

**public decimal Salary { get; set; }**

***// Additional Information***

**public string Address { get; set; }**

**public string Comments { get; set; }**

***// Boolean fields (checkboxes)***

**public bool IsActive { get; set; }**

**public bool ReceiveNewsletter { get; set; }**

***// Radio button selections***

**public string MaritalStatus { get; set; } *// "Single", "Married", "Divorced"***

**public string EmploymentType { get; set; } *// "Full-time", "Part-time", "Contract"***

***// Dropdown selections***

**public string Country { get; set; }**

**public string State { get; set; }**

***// Metadata***

**public DateTime RetrievedAt { get; set; } = DateTime.Now;**

**public string PageUrl { get; set; }**

***// Constructor***

**public EmployeeFormData()**

**{**

***// Initialize with default values if needed***

**IsActive = true;**

**ReceiveNewsletter = false;**

**}**

***// Validation method***

**public bool IsValid()**

**{**

**return !string.IsNullOrEmpty(FirstName) &&**

**!string.IsNullOrEmpty(LastName) &&**

**!string.IsNullOrEmpty(Email) &&**

**IsValidEmail(Email);**

**}**

**private bool IsValidEmail(string email)**

**{**

**try**

**{**

**var addr = new System.Net.Mail.MailAddress(email);**

**return addr.Address == email;**

**}**

**catch**

**{**

**return false;**

**}**

**}**

***// Comparison method***

**public override bool Equals(object obj)**

**{**

**if (obj is EmployeeFormData other)**

**{**

**return FirstName == other.FirstName &&**

**LastName == other.LastName &&**

**Email == other.Email &&**

**PhoneNumber == other.PhoneNumber &&**

**EmployeeId == other.EmployeeId &&**

**Department == other.Department &&**

**JobTitle == other.JobTitle &&**

**IsActive == other.IsActive;**

**}**

**return false;**

**}**

**public override int GetHashCode()**

**{**

**return HashCode.Combine(FirstName, LastName, Email, EmployeeId);**

**}**

**}**

**}**

**2. Advanced Data Model with Attributes**

**csharp**

**using System;**

**using System.ComponentModel;**

**using System.ComponentModel.DataAnnotations;**

**namespace YourProject.Models**

**{**

**public class AdvancedEmployeeFormData**

**{**

**[Required]**

**[DisplayName("First Name")]**

**[StringLength(50)]**

**public string FirstName { get; set; }**

**[Required]**

**[DisplayName("Last Name")]**

**[StringLength(50)]**

**public string LastName { get; set; }**

**[Required]**

**[EmailAddress]**

**[DisplayName("Email Address")]**

**public string Email { get; set; }**

**[Phone]**

**[DisplayName("Phone Number")]**

**public string PhoneNumber { get; set; }**

**[Required]**

**[DisplayName("Employee ID")]**

**[RegularExpression(@"^EMP\d{6}$", ErrorMessage = "Employee ID must be in format EMP123456")]**

**public string EmployeeId { get; set; }**

**[Required]**

**[DisplayName("Department")]**

**public string Department { get; set; }**

**[Required]**

**[DisplayName("Job Title")]**

**public string JobTitle { get; set; }**

**[Required]**

**[DataType(DataType.Date)]**

**[DisplayName("Start Date")]**

**public DateTime StartDate { get; set; }**

**[Required]**

**[Range(0.01, 1000000.00)]**

**[DisplayName("Annual Salary")]**

**public decimal Salary { get; set; }**

**[MaxLength(200)]**

**[DisplayName("Address")]**

**public string Address { get; set; }**

**[MaxLength(500)]**

**[DisplayName("Comments")]**

**public string Comments { get; set; }**

**[DisplayName("Active Employee")]**

**public bool IsActive { get; set; }**

**[DisplayName("Receive Newsletter")]**

**public bool ReceiveNewsletter { get; set; }**

**[DisplayName("Marital Status")]**

**public string MaritalStatus { get; set; }**

**[Required]**

**[DisplayName("Employment Type")]**

**public string EmploymentType { get; set; }**

**[Required]**

**[DisplayName("Country")]**

**public string Country { get; set; }**

**[DisplayName("State")]**

**public string State { get; set; }**

***// Metadata properties***

**public DateTime RetrievedAt { get; set; } = DateTime.Now;**

**public string PageUrl { get; set; }**

**public string TestRunId { get; set; }**

***// Validation context***

**public List<ValidationResult> ValidationErrors { get; set; } = new List<ValidationResult>();**

**public bool ValidateModel()**

**{**

**var context = new ValidationContext(this);**

**ValidationErrors.Clear();**

**return Validator.TryValidateObject(this, context, ValidationErrors, true);**

**}**

**}**

**}**

**Page Object Implementation**

**1. Base Page Object Class**

**csharp**

**using OpenQA.Selenium;**

**using OpenQA.Selenium.Support.UI;**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**namespace YourProject.Pages**

**{**

**public abstract class BasePage**

**{**

**protected IWebDriver Driver;**

**protected WebDriverWait Wait;**

**public BasePage(IWebDriver driver)**

**{**

**Driver = driver;**

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

**}**

***// Helper methods for data extraction***

**protected string GetTextValue(By locator)**

**{**

**try**

**{**

**var element = Wait.Until(d => d.FindElement(locator));**

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

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine($"Error getting text value for {locator}: {ex.Message}");**

**return "";**

**}**

**}**

**protected bool GetCheckboxValue(By locator)**

**{**

**try**

**{**

**var element = Wait.Until(d => d.FindElement(locator));**

**return element.Selected;**

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine($"Error getting checkbox value for {locator}: {ex.Message}");**

**return false;**

**}**

**}**

**protected string GetSelectedDropdownValue(By locator)**

**{**

**try**

**{**

**var element = Wait.Until(d => d.FindElement(locator));**

**var select = new SelectElement(element);**

**return select.SelectedOption.Text;**

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine($"Error getting dropdown value for {locator}: {ex.Message}");**

**return "";**

**}**

**}**

**protected string GetSelectedRadioValue(string name)**

**{**

**try**

**{**

**var selectedRadio = Driver.FindElement(By.CssSelector($"input[name='{name}']:checked"));**

**return selectedRadio.GetAttribute("value");**

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine($"Error getting radio value for name '{name}': {ex.Message}");**

**return "";**

**}**

**}**

**protected string GetTextareaValue(By locator)**

**{**

**try**

**{**

**var element = Wait.Until(d => d.FindElement(locator));**

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

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine($"Error getting textarea value for {locator}: {ex.Message}");**

**return "";**

**}**

**}**

**}**

**}**

**2. Employee Form Page Object**

**csharp**

**using OpenQA.Selenium;**

**using YourProject.Models;**

**using System;**

**using System.Globalization;**

**namespace YourProject.Pages**

**{**

**public class EmployeeFormPage : BasePage**

**{**

***// Element locators***

**private readonly By \_firstNameInput = By.Id("firstName");**

**private readonly By \_lastNameInput = By.Id("lastName");**

**private readonly By \_emailInput = By.Id("email");**

**private readonly By \_phoneInput = By.Id("phone");**

**private readonly By \_employeeIdInput = By.Id("employeeId");**

**private readonly By \_departmentDropdown = By.Id("department");**

**private readonly By \_jobTitleInput = By.Id("jobTitle");**

**private readonly By \_startDateInput = By.Id("startDate");**

**private readonly By \_salaryInput = By.Id("salary");**

**private readonly By \_addressTextarea = By.Id("address");**

**private readonly By \_commentsTextarea = By.Id("comments");**

**private readonly By \_isActiveCheckbox = By.Id("isActive");**

**private readonly By \_receiveNewsletterCheckbox = By.Id("receiveNewsletter");**

**private readonly By \_countryDropdown = By.Id("country");**

**private readonly By \_stateDropdown = By.Id("state");**

***// Radio button names***

**private const string MaritalStatusRadioName = "maritalStatus";**

**private const string EmploymentTypeRadioName = "employmentType";**

**public EmployeeFormPage(IWebDriver driver) : base(driver)**

**{**

**}**

***// Form interaction methods***

**public void FillFirstName(string firstName)**

**{**

**var element = Wait.Until(d => d.FindElement(\_firstNameInput));**

**element.Clear();**

**element.SendKeys(firstName);**

**}**

**public void FillLastName(string lastName)**

**{**

**var element = Wait.Until(d => d.FindElement(\_lastNameInput));**

**element.Clear();**

**element.SendKeys(lastName);**

**}**

**public void FillEmail(string email)**

**{**

**var element = Wait.Until(d => d.FindElement(\_emailInput));**

**element.Clear();**

**element.SendKeys(email);**

**}**

**public void SelectDepartment(string department)**

**{**

**var element = Wait.Until(d => d.FindElement(\_departmentDropdown));**

**var select = new SelectElement(element);**

**select.SelectByText(department);**

**}**

**public void SetActiveStatus(bool isActive)**

**{**

**var checkbox = Wait.Until(d => d.FindElement(\_isActiveCheckbox));**

**if (checkbox.Selected != isActive)**

**{**

**checkbox.Click();**

**}**

**}**

**public void SelectMaritalStatus(string status)**

**{**

**var radioButton = Driver.FindElement(By.CssSelector($"input[name='{MaritalStatusRadioName}'][value='{status}']"));**

**radioButton.Click();**

**}**

**public void SubmitForm()**

**{**

**var submitButton = Wait.Until(d => d.FindElement(By.Id("submitButton")));**

**submitButton.Click();**

**}**

***// Validation methods***

**public bool IsFormDisplayed()**

**{**

**try**

**{**

**Wait.Until(d => d.FindElement(\_firstNameInput));**

**return true;**

**}**

**catch**

**{**

**return false;**

**}**

**}**

**public List<string> GetValidationErrors()**

**{**

**var errors = new List<string>();**

**try**

**{**

**var errorElements = Driver.FindElements(By.CssSelector(".validation-error, .error-message"));**

**foreach (var element in errorElements)**

**{**

**if (!string.IsNullOrEmpty(element.Text))**

**{**

**errors.Add(element.Text);**

**}**

**}**

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine($"Error retrieving validation errors: {ex.Message}");**

**}**

**return errors;**

**}**

**}**

**}**

**GetPageData Function**

**1. Complete GetPageData Implementation**

**csharp**

**public partial class EmployeeFormPage : BasePage**

**{**

***/// <summary>***

***/// Retrieves all current form data and returns it as an EmployeeFormData model***

***/// </summary>***

***/// <returns>EmployeeFormData containing all current form values</returns>***

**public EmployeeFormData GetPageData()**

**{**

**try**

**{**

**var formData = new EmployeeFormData**

**{**

***// Personal Information***

**FirstName = GetTextValue(\_firstNameInput),**

**LastName = GetTextValue(\_lastNameInput),**

**Email = GetTextValue(\_emailInput),**

**PhoneNumber = GetTextValue(\_phoneInput),**

***// Employment Details***

**EmployeeId = GetTextValue(\_employeeIdInput),**

**Department = GetSelectedDropdownValue(\_departmentDropdown),**

**JobTitle = GetTextValue(\_jobTitleInput),**

**StartDate = GetTextValue(\_startDateInput),**

***// Additional Information***

**Address = GetTextareaValue(\_addressTextarea),**

**Comments = GetTextareaValue(\_commentsTextarea),**

***// Boolean fields***

**IsActive = GetCheckboxValue(\_isActiveCheckbox),**

**ReceiveNewsletter = GetCheckboxValue(\_receiveNewsletterCheckbox),**

***// Radio button selections***

**MaritalStatus = GetSelectedRadioValue(MaritalStatusRadioName),**

**EmploymentType = GetSelectedRadioValue(EmploymentTypeRadioName),**

***// Dropdown selections***

**Country = GetSelectedDropdownValue(\_countryDropdown),**

**State = GetSelectedDropdownValue(\_stateDropdown),**

***// Metadata***

**PageUrl = Driver.Url,**

**RetrievedAt = DateTime.Now**

**};**

***// Parse salary if present***

**var salaryText = GetTextValue(\_salaryInput);**

**if (decimal.TryParse(salaryText.Replace("$", "").Replace(",", ""), out decimal salary))**

**{**

**formData.Salary = salary;**

**}**

**return formData;**

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine($"Error retrieving form data: {ex.Message}");**

**throw new Exception($"Failed to retrieve form data: {ex.Message}", ex);**

**}**

**}**

***/// <summary>***

***/// Retrieves form data with retry mechanism for flaky elements***

***/// </summary>***

***/// <param name="maxRetries">Maximum number of retry attempts</param>***

***/// <returns>EmployeeFormData or null if all retries fail</returns>***

**public EmployeeFormData GetPageDataWithRetry(int maxRetries = 3)**

**{**

**Exception lastException = null;**

**for (int attempt = 1; attempt <= maxRetries; attempt++)**

**{**

**try**

**{**

**return GetPageData();**

**}**

**catch (Exception ex)**

**{**

**lastException = ex;**

**Console.WriteLine($"Attempt {attempt} failed to retrieve form data: {ex.Message}");**

**if (attempt < maxRetries)**

**{**

**System.Threading.Thread.Sleep(1000); *// Wait 1 second before retry***

**}**

**}**

**}**

**throw new Exception($"Failed to retrieve form data after {maxRetries} attempts", lastException);**

**}**

***/// <summary>***

***/// Retrieves specific field values with validation***

***/// </summary>***

***/// <param name="fieldNames">List of field names to retrieve</param>***

***/// <returns>Dictionary containing field names and their values</returns>***

**public Dictionary<string, object> GetSpecificFields(List<string> fieldNames)**

**{**

**var fieldData = new Dictionary<string, object>();**

**foreach (var fieldName in fieldNames)**

**{**

**try**

**{**

**object value = fieldName.ToLower() switch**

**{**

**"firstname" => GetTextValue(\_firstNameInput),**

**"lastname" => GetTextValue(\_lastNameInput),**

**"email" => GetTextValue(\_emailInput),**

**"phone" => GetTextValue(\_phoneInput),**

**"employeeid" => GetTextValue(\_employeeIdInput),**

**"department" => GetSelectedDropdownValue(\_departmentDropdown),**

**"jobtitle" => GetTextValue(\_jobTitleInput),**

**"startdate" => GetTextValue(\_startDateInput),**

**"salary" => GetTextValue(\_salaryInput),**

**"address" => GetTextareaValue(\_addressTextarea),**

**"comments" => GetTextareaValue(\_commentsTextarea),**

**"isactive" => GetCheckboxValue(\_isActiveCheckbox),**

**"receivenewsletter" => GetCheckboxValue(\_receiveNewsletterCheckbox),**

**"maritalstatus" => GetSelectedRadioValue(MaritalStatusRadioName),**

**"employmenttype" => GetSelectedRadioValue(EmploymentTypeRadioName),**

**"country" => GetSelectedDropdownValue(\_countryDropdown),**

**"state" => GetSelectedDropdownValue(\_stateDropdown),**

**\_ => null**

**};**

**if (value != null)**

**{**

**fieldData[fieldName] = value;**

**}**

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine($"Error retrieving field '{fieldName}': {ex.Message}");**

**fieldData[fieldName] = null;**

**}**

**}**

**return fieldData;**

**}**

***/// <summary>***

***/// Compares current form data with expected data***

***/// </summary>***

***/// <param name="expectedData">Expected form data</param>***

***/// <returns>Comparison result with differences</returns>***

**public FormDataComparisonResult CompareFormData(EmployeeFormData expectedData)**

**{**

**var currentData = GetPageData();**

**var result = new FormDataComparisonResult**

**{**

**IsMatch = true,**

**CurrentData = currentData,**

**ExpectedData = expectedData,**

**Differences = new List<FieldDifference>()**

**};**

***// Compare each field***

**CompareField(result, nameof(EmployeeFormData.FirstName), currentData.FirstName, expectedData.FirstName);**

**CompareField(result, nameof(EmployeeFormData.LastName), currentData.LastName, expectedData.LastName);**

**CompareField(result, nameof(EmployeeFormData.Email), currentData.Email, expectedData.Email);**

**CompareField(result, nameof(EmployeeFormData.PhoneNumber), currentData.PhoneNumber, expectedData.PhoneNumber);**

**CompareField(result, nameof(EmployeeFormData.EmployeeId), currentData.EmployeeId, expectedData.EmployeeId);**

**CompareField(result, nameof(EmployeeFormData.Department), currentData.Department, expectedData.Department);**

**CompareField(result, nameof(EmployeeFormData.JobTitle), currentData.JobTitle, expectedData.JobTitle);**

**CompareField(result, nameof(EmployeeFormData.IsActive), currentData.IsActive.ToString(), expectedData.IsActive.ToString());**

**CompareField(result, nameof(EmployeeFormData.MaritalStatus), currentData.MaritalStatus, expectedData.MaritalStatus);**

**return result;**

**}**

**private void CompareField(FormDataComparisonResult result, string fieldName, string currentValue, string expectedValue)**

**{**

**if (currentValue != expectedValue)**

**{**

**result.IsMatch = false;**

**result.Differences.Add(new FieldDifference**

**{**

**FieldName = fieldName,**

**CurrentValue = currentValue ?? "",**

**ExpectedValue = expectedValue ?? ""**

**});**

**}**

**}**

**}**

***// Supporting classes for comparison***

**public class FormDataComparisonResult**

**{**

**public bool IsMatch { get; set; }**

**public EmployeeFormData CurrentData { get; set; }**

**public EmployeeFormData ExpectedData { get; set; }**

**public List<FieldDifference> Differences { get; set; }**

**public override string ToString()**

**{**

**if (IsMatch)**

**return "Form data matches expected values.";**

**var diff = string.Join(", ", Differences.Select(d =>**

**$"{d.FieldName}: '{d.CurrentValue}' != '{d.ExpectedValue}'"));**

**return $"Form data differences: {diff}";**

**}**

**}**

**public class FieldDifference**

**{**

**public string FieldName { get; set; }**

**public string CurrentValue { get; set; }**

**public string ExpectedValue { get; set; }**

**}**

**Usage in Tests**

**1. Basic Test Implementation**

**csharp**

**using NUnit.Framework;**

**using OpenQA.Selenium;**

**using OpenQA.Selenium.Chrome;**

**using YourProject.Pages;**

**using YourProject.Models;**

**using System;**

**namespace YourProject.Tests**

**{**

**[TestFixture]**

**public class EmployeeFormTests**

**{**

**private IWebDriver \_driver;**

**private EmployeeFormPage \_employeeFormPage;**

**[SetUp]**

**public void Setup()**

**{**

**\_driver = new ChromeDriver();**

**\_driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);**

**\_employeeFormPage = new EmployeeFormPage(\_driver);**

**}**

**[TearDown]**

**public void TearDown()**

**{**

**\_driver?.Quit();**

**}**

**[Test]**

**public void TestFormDataRetrieval\_WhenFormIsEmpty\_ShouldReturnEmptyValues()**

**{**

***// Arrange***

**\_driver.Navigate().GoToUrl("https://example.com/employee-form");**

***// Act***

**var formData = \_employeeFormPage.GetPageData();**

***// Assert***

**Assert.That(formData, Is.Not.Null);**

**Assert.That(formData.FirstName, Is.Empty);**

**Assert.That(formData.LastName, Is.Empty);**

**Assert.That(formData.Email, Is.Empty);**

**Assert.That(formData.IsActive, Is.False);**

**}**

**[Test]**

**public void TestFormDataRetrieval\_AfterFillingForm\_ShouldReturnCorrectValues()**

**{**

***// Arrange***

**\_driver.Navigate().GoToUrl("https://example.com/employee-form");**

**var expectedData = new EmployeeFormData**

**{**

**FirstName = "John",**

**LastName = "Doe",**

**Email = "john.doe@company.com",**

**Department = "Engineering",**

**IsActive = true,**

**MaritalStatus = "Married"**

**};**

***// Act - Fill the form***

**\_employeeFormPage.FillFirstName(expectedData.FirstName);**

**\_employeeFormPage.FillLastName(expectedData.LastName);**

**\_employeeFormPage.FillEmail(expectedData.Email);**

**\_employeeFormPage.SelectDepartment(expectedData.Department);**

**\_employeeFormPage.SetActiveStatus(expectedData.IsActive);**

**\_employeeFormPage.SelectMaritalStatus(expectedData.MaritalStatus);**

***// Act - Retrieve the data***

**var actualData = \_employeeFormPage.GetPageData();**

***// Assert***

**Assert.That(actualData.FirstName, Is.EqualTo(expectedData.FirstName));**

**Assert.That(actualData.LastName, Is.EqualTo(expectedData.LastName));**

**Assert.That(actualData.Email, Is.EqualTo(expectedData.Email));**

**Assert.That(actualData.Department, Is.EqualTo(expectedData.Department));**

**Assert.That(actualData.IsActive, Is.EqualTo(expectedData.IsActive));**

**Assert.That(actualData.MaritalStatus, Is.EqualTo(expectedData.MaritalStatus));**

**}**

**[Test]**

**public void TestFormDataComparison\_WithMatchingData\_ShouldReturnTrue()**

**{**

***// Arrange***

**\_driver.Navigate().GoToUrl("https://example.com/employee-form");**

**var testData = new EmployeeFormData**

**{**

**FirstName = "Jane",**

**LastName = "Smith",**

**Email = "jane.smith@company.com",**

**Department = "Marketing",**

**IsActive = true**

**};**

***// Fill form with test data***

**FillFormWithData(testData);**

***// Act***

**var comparisonResult = \_employeeFormPage.CompareFormData(testData);**

***// Assert***

**Assert.That(comparisonResult.IsMatch, Is.True);**

**Assert.That(comparisonResult.Differences, Is.Empty);**

**}**

**[Test]**

**public void TestFormDataComparison\_WithDifferentData\_ShouldReturnFalseWithDifferences()**

**{**

***// Arrange***

**\_driver.Navigate().GoToUrl("https://example.com/employee-form");**

**var filledData = new EmployeeFormData**

**{**

**FirstName = "John",**

**LastName = "Doe",**

**Email = "john.doe@company.com"**

**};**

**var expectedData = new EmployeeFormData**

**{**

**FirstName = "Jane",**

**LastName = "Smith",**

**Email = "jane.smith@company.com"**

**};**

***// Fill form with different data***

**FillFormWithData(filledData);**

***// Act***

**var comparisonResult = \_employeeFormPage.CompareFormData(expectedData);**

***// Assert***

**Assert.That(comparisonResult.IsMatch, Is.False);**

**Assert.That(comparisonResult.Differences.Count, Is.EqualTo(3));**

**Assert.That(comparisonResult.Differences[0].FieldName, Is.EqualTo("FirstName"));**

**Assert.That(comparisonResult.Differences[0].CurrentValue, Is.EqualTo("John"));**

**Assert.That(comparisonResult.Differences[0].ExpectedValue, Is.EqualTo("Jane"));**

**}**

**private void FillFormWithData(EmployeeFormData data)**

**{**

**if (!string.IsNullOrEmpty(data.FirstName))**

**\_employeeFormPage.FillFirstName(data.FirstName);**

**if (!string.IsNullOrEmpty(data.LastName))**

**\_employeeFormPage.FillLastName(data.LastName);**

**if (!string.IsNullOrEmpty(data.Email))**

**\_employeeFormPage.FillEmail(data.Email);**

**if (!string.IsNullOrEmpty(data.Department))**

**\_employeeFormPage.SelectDepartment(data.Department);**

**\_employeeFormPage.SetActiveStatus(data.IsActive);**

**if (!string.IsNullOrEmpty(data.MaritalStatus))**

**\_employeeFormPage.SelectMaritalStatus(data.MaritalStatus);**

**}**

**}**

**}**

**2. Advanced Test Scenarios**

**csharp**

**[TestFixture]**

**public class AdvancedEmployeeFormTests**

**{**

**private IWebDriver \_driver;**

**private EmployeeFormPage \_employeeFormPage;**

**private TestDataManager \_testDataManager;**

**[SetUp]**

**public void Setup()**

**{**

**\_driver = new ChromeDriver();**

**\_employeeFormPage = new EmployeeFormPage(\_driver);**

**\_testDataManager = new TestDataManager();**

**}**

**[Test]**

**public void TestFormDataPersistence\_BeforeAndAfterPageReload()**

**{**

***// Arrange***

**\_driver.Navigate().GoToUrl("https://example.com/employee-form");**

**var testData = \_testDataManager.GetValidEmployeeData();**

***// Fill form and capture data***

**FillFormWithData(testData);**

**var dataBeforeReload = \_employeeFormPage.GetPageData();**

***// Act - Reload page***

**\_driver.Navigate().Refresh();**

**var dataAfterReload = \_employeeFormPage.GetPageData();**

***// Assert - Data should be preserved if form has auto-save***

**var comparison = \_employeeFormPage.CompareFormData(dataBeforeReload);**

**if (HasAutoSaveFeature())**

**{**

**Assert.That(comparison.IsMatch, Is.True, "Form data should persist after page reload");**

**}**

**else**

**{**

**Assert.That(dataAfterReload.FirstName, Is.Empty, "Form should be empty after reload without auto-save");**

**}**

**}**

**[Test]**

**public void TestFormDataValidation\_WithInvalidData\_ShouldShowErrors()**

**{**

***// Arrange***

**\_driver.Navigate().GoToUrl("https://example.com/employee-form");**

**var invalidData = new EmployeeFormData**

**{**

**FirstName = "", *// Required field empty***

**Email = "invalid-email", *// Invalid email format***

**Salary = -1000 *// Invalid salary***

**};**

***// Act***

**FillFormWithData(invalidData);**

**\_employeeFormPage.SubmitForm();**

**var retrievedData = \_employeeFormPage.GetPageData();**

**var validationErrors = \_employeeFormPage.GetValidationErrors();**

***// Assert***

**Assert.That(validationErrors.Count, Is.GreaterThan(0));**

**Assert.That(validationErrors.Any(e => e.Contains("First Name")), Is.True);**

**Assert.That(validationErrors.Any(e => e.Contains("email")), Is.True);**

**}**

**[Test]**

**[TestCaseSource(nameof(GetTestDataVariations))]**

**public void TestFormDataRetrieval\_WithVariousDataSets(EmployeeFormData testData, string scenario)**

**{**

***// Arrange***

**\_driver.Navigate().GoToUrl("https://example.com/employee-form");**

***// Act***

**FillFormWithData(testData);**

**var retrievedData = \_employeeFormPage.GetPageData();**

***// Assert***

**var comparison = \_employeeFormPage.CompareFormData(testData);**

**Assert.That(comparison.IsMatch, Is.True,**

**$"Scenario '{scenario}' failed: {comparison}");**

***// Additional validations based on scenario***

**switch (scenario)**

**{**

**case "MinimalData":**

**Assert.That(retrievedData.IsValid(), Is.True);**

**break;**

**case "CompleteData":**

**Assert.That(string.IsNullOrEmpty(retrievedData.Comments), Is.False);**

**Assert.That(retrievedData.Salary, Is.GreaterThan(0));**

**break;**

**case "SpecialCharacters":**

**Assert.That(retrievedData.FirstName, Contains.Substring("'"));**

**break;**

**}**

**}**

**public static IEnumerable<TestCaseData> GetTestDataVariations()**

**{**

**yield return new TestCaseData(**

**new EmployeeFormData**

**{**

**FirstName = "John",**

**LastName = "Doe",**

**Email = "john.doe@company.com",**

**IsActive = true**

**},**

**"MinimalData"**

**);**

**yield return new TestCaseData(**

**new EmployeeFormData**

**{**

**FirstName = "Jane",**

**LastName = "Smith",**

**Email = "jane.smith@company.com",**

**PhoneNumber = "(555) 123-4567",**

**EmployeeId = "EMP123456",**

**Department = "Engineering",**

**JobTitle = "Senior Developer",**

**Salary = 85000,**

**Address = "123 Main St, City, State 12345",**

**Comments = "Experienced developer with strong technical skills",**

**IsActive = true,**

**ReceiveNewsletter = true,**

**MaritalStatus = "Single",**

**EmploymentType = "Full-time",**

**Country = "United States",**

**State = "California"**

**},**

**"CompleteData"**

**);**

**yield return new TestCaseData(**

**new EmployeeFormData**

**{**

**FirstName = "Mary-Jane",**

**LastName = "O'Connor",**

**Email = "mary.jane+test@company-name.com",**

**Comments = "Employee with special characters: @#$%^&\*()",**

**IsActive = false**

**},**

**"SpecialCharacters"**

**);**

**}**

**[Test]**

**public void TestFormDataRetrieval\_WithRetryMechanism\_ShouldHandleTransientFailures()**

**{**

***// Arrange***

**\_driver.Navigate().GoToUrl("https://example.com/slow-loading-form");**

**var testData = \_testDataManager.GetValidEmployeeData();**

***// Act & Assert***

**Assert.DoesNotThrow(() =>**

**{**

**var retrievedData = \_employeeFormPage.GetPageDataWithRetry(maxRetries: 5);**

**Assert.That(retrievedData, Is.Not.Null);**

**});**

**}**

**[Test]**

**public void TestSpecificFieldRetrieval\_ShouldReturnOnlyRequestedFields()**

**{**

***// Arrange***

**\_driver.Navigate().GoToUrl("https://example.com/employee-form");**

**var testData = \_testDataManager.GetValidEmployeeData();**

**FillFormWithData(testData);**

**var requestedFields = new List<string> { "FirstName", "LastName", "Email" };**

***// Act***

**var fieldData = \_employeeFormPage.GetSpecificFields(requestedFields);**

***// Assert***

**Assert.That(fieldData.Count, Is.EqualTo(3));**

**Assert.That(fieldData.ContainsKey("FirstName"), Is.True);**

**Assert.That(fieldData.ContainsKey("LastName"), Is.True);**

**Assert.That(fieldData.ContainsKey("Email"), Is.True);**

**Assert.That(fieldData["FirstName"].ToString(), Is.EqualTo(testData.FirstName));**

**}**

**private bool HasAutoSaveFeature()**

**{**

***// Implementation to check if the form has auto-save functionality***

**try**

**{**

**\_driver.FindElement(By.CssSelector("[data-auto-save='true']"));**

**return true;**

**}**

**catch (NoSuchElementException)**

**{**

**return false;**

**}**

**}**

**}**

***// Test Data Manager Helper Class***

**public class TestDataManager**

**{**

**private readonly Random \_random = new Random();**

**public EmployeeFormData GetValidEmployeeData()**

**{**

**return new EmployeeFormData**

**{**

**FirstName = $"TestUser{\_random.Next(1000)}",**

**LastName = $"LastName{\_random.Next(1000)}",**

**Email = $"test{\_random.Next(1000)}@company.com",**

**PhoneNumber = $"555-{\_random.Next(100, 999)}-{\_random.Next(1000, 9999)}",**

**EmployeeId = $"EMP{\_random.Next(100000, 999999)}",**

**Department = "QA Testing",**

**JobTitle = "Test Engineer",**

**Salary = \_random.Next(50000, 100000),**

**IsActive = true,**

**MaritalStatus = "Single",**

**EmploymentType = "Full-time",**

**Country = "United States"**

**};**

**}**

**public EmployeeFormData GetInvalidEmployeeData()**

**{**

**return new EmployeeFormData**

**{**

**FirstName = "", *// Invalid - empty required field***

**Email = "invalid.email", *// Invalid - no @ symbol***

**Salary = -1000, *// Invalid - negative salary***

**PhoneNumber = "123", *// Invalid - too short***

**EmployeeId = "INVALID" *// Invalid - wrong format***

**};**

**}**

**public List<EmployeeFormData> GetBulkTestData(int count)**

**{**

**var testDataList = new List<EmployeeFormData>();**

**for (int i = 0; i < count; i++)**

**{**

**testDataList.Add(GetValidEmployeeData());**

**}**

**return testDataList;**

**}**

**}**

**Best Practices**

**1. Error Handling and Resilience**

**csharp**

**public class RobustFormDataRetrieval**

**{**

**private readonly IWebDriver \_driver;**

**private readonly ILogger \_logger;**

**public RobustFormDataRetrieval(IWebDriver driver, ILogger logger)**

**{**

**\_driver = driver;**

**\_logger = logger;**

**}**

***/// <summary>***

***/// Enhanced GetPageData with comprehensive error handling***

***/// </summary>***

**public EmployeeFormData GetPageDataSafely()**

**{**

**var formData = new EmployeeFormData();**

**var errors = new List<string>();**

***// Use reflection to get all properties***

**var properties = typeof(EmployeeFormData).GetProperties()**

**.Where(p => p.CanWrite && p.PropertyType == typeof(string) || p.PropertyType == typeof(bool));**

**foreach (var property in properties)**

**{**

**try**

**{**

**var value = ExtractFieldValueSafely(property.Name);**

**if (value != null)**

**{**

**property.SetValue(formData, value);**

**}**

**}**

**catch (Exception ex)**

**{**

**errors.Add($"Failed to extract {property.Name}: {ex.Message}");**

**\_logger.LogWarning($"Failed to extract field {property.Name}: {ex.Message}");**

**}**

**}**

**if (errors.Any())**

**{**

**\_logger.LogWarning($"Form data extraction completed with {errors.Count} errors");**

**}**

**return formData;**

**}**

**private object ExtractFieldValueSafely(string fieldName)**

**{**

**var maxRetries = 3;**

**var delay = TimeSpan.FromMilliseconds(500);**

**for (int attempt = 1; attempt <= maxRetries; attempt++)**

**{**

**try**

**{**

**return fieldName.ToLower() switch**

**{**

**"firstname" => GetTextValueWithWait(By.Id("firstName")),**

**"lastname" => GetTextValueWithWait(By.Id("lastName")),**

**"email" => GetTextValueWithWait(By.Id("email")),**

**"isactive" => GetCheckboxValueWithWait(By.Id("isActive")),**

**"department" => GetDropdownValueWithWait(By.Id("department")),**

**\_ => null**

**};**

**}**

**catch (StaleElementReferenceException)**

**{**

**if (attempt < maxRetries)**

**{**

**Thread.Sleep(delay);**

**continue;**

**}**

**throw;**

**}**

**catch (NoSuchElementException ex)**

**{**

**\_logger.LogDebug($"Element not found for field {fieldName}: {ex.Message}");**

**return null;**

**}**

**}**

**return null;**

**}**

**private string GetTextValueWithWait(By locator, int timeoutSeconds = 5)**

**{**

**var wait = new WebDriverWait(\_driver, TimeSpan.FromSeconds(timeoutSeconds));**

**var element = wait.Until(driver =>**

**{**

**try**

**{**

**var el = driver.FindElement(locator);**

**return el.Displayed && el.Enabled ? el : null;**

**}**

**catch (StaleElementReferenceException)**

**{**

**return null;**

**}**

**});**

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

**}**

**private bool GetCheckboxValueWithWait(By locator, int timeoutSeconds = 5)**

**{**

**var wait = new WebDriverWait(\_driver, TimeSpan.FromSeconds(timeoutSeconds));**

**var element = wait.Until(driver =>**

**{**

**try**

**{**

**var el = driver.FindElement(locator);**

**return el.Displayed ? el : null;**

**}**

**catch (StaleElementReferenceException)**

**{**

**return null;**

**}**

**});**

**return element?.Selected ?? false;**

**}**

**private string GetDropdownValueWithWait(By locator, int timeoutSeconds = 5)**

**{**

**var wait = new WebDriverWait(\_driver, TimeSpan.FromSeconds(timeoutSeconds));**

**var element = wait.Until(driver =>**

**{**

**try**

**{**

**var el = driver.FindElement(locator);**

**return el.Displayed && el.Enabled ? el : null;**

**}**

**catch (StaleElementReferenceException)**

**{**

**return null;**

**}**

**});**

**if (element != null)**

**{**

**var select = new SelectElement(element);**

**return select.SelectedOption?.Text ?? "";**

**}**

**return "";**

**}**

**}**

**2. Configuration Management**

**csharp**

***// FormFieldConfiguration.cs***

**public class FormFieldConfiguration**

**{**

**public Dictionary<string, By> FieldLocators { get; set; }**

**public Dictionary<string, string> RadioGroupNames { get; set; }**

**public List<string> RequiredFields { get; set; }**

**public Dictionary<string, Func<string, object>> FieldTransformers { get; set; }**

**public static FormFieldConfiguration GetDefaultConfiguration()**

**{**

**return new FormFieldConfiguration**

**{**

**FieldLocators = new Dictionary<string, By>**

**{**

**["FirstName"] = By.Id("firstName"),**

**["LastName"] = By.Id("lastName"),**

**["Email"] = By.Id("email"),**

**["Phone"] = By.Id("phone"),**

**["Department"] = By.Id("department"),**

**["IsActive"] = By.Id("isActive"),**

**["Comments"] = By.Id("comments")**

**},**

**RadioGroupNames = new Dictionary<string, string>**

**{**

**["MaritalStatus"] = "maritalStatus",**

**["EmploymentType"] = "employmentType"**

**},**

**RequiredFields = new List<string>**

**{**

**"FirstName", "LastName", "Email", "Department"**

**},**

**FieldTransformers = new Dictionary<string, Func<string, object>>**

**{**

**["Salary"] = text => decimal.TryParse(text.Replace("$", "").Replace(",", ""), out var result) ? result : 0m,**

**["StartDate"] = text => DateTime.TryParse(text, out var date) ? date : DateTime.MinValue,**

**["Phone"] = text => System.Text.RegularExpressions.Regex.Replace(text, @"[^\d]", "")**

**}**

**};**

**}**

**}**

***// ConfigurableFormPage.cs***

**public class ConfigurableFormPage : BasePage**

**{**

**private readonly FormFieldConfiguration \_configuration;**

**public ConfigurableFormPage(IWebDriver driver, FormFieldConfiguration configuration = null)**

**: base(driver)**

**{**

**\_configuration = configuration ?? FormFieldConfiguration.GetDefaultConfiguration();**

**}**

**public EmployeeFormData GetPageData()**

**{**

**var formData = new EmployeeFormData();**

**var properties = typeof(EmployeeFormData).GetProperties();**

**foreach (var property in properties)**

**{**

**if (\_configuration.FieldLocators.TryGetValue(property.Name, out var locator))**

**{**

**var value = GetFieldValue(property.Name, locator, property.PropertyType);**

**if (value != null)**

**{**

**property.SetValue(formData, value);**

**}**

**}**

**}**

**formData.RetrievedAt = DateTime.Now;**

**formData.PageUrl = Driver.Url;**

**return formData;**

**}**

**private object GetFieldValue(string fieldName, By locator, Type propertyType)**

**{**

**try**

**{**

**string rawValue;**

**if (propertyType == typeof(bool))**

**{**

**return GetCheckboxValue(locator);**

**}**

**else if (\_configuration.RadioGroupNames.ContainsKey(fieldName))**

**{**

**rawValue = GetSelectedRadioValue(\_configuration.RadioGroupNames[fieldName]);**

**}**

**else if (IsDropdownElement(locator))**

**{**

**rawValue = GetSelectedDropdownValue(locator);**

**}**

**else**

**{**

**rawValue = GetTextValue(locator);**

**}**

***// Apply transformation if configured***

**if (\_configuration.FieldTransformers.TryGetValue(fieldName, out var transformer))**

**{**

**return transformer(rawValue);**

**}**

**return rawValue;**

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine($"Error extracting field '{fieldName}': {ex.Message}");**

**return GetDefaultValueForType(propertyType);**

**}**

**}**

**private bool IsDropdownElement(By locator)**

**{**

**try**

**{**

**var element = Driver.FindElement(locator);**

**return element.TagName.ToLower() == "select";**

**}**

**catch**

**{**

**return false;**

**}**

**}**

**private object GetDefaultValueForType(Type type)**

**{**

**if (type == typeof(string)) return "";**

**if (type == typeof(bool)) return false;**

**if (type == typeof(decimal)) return 0m;**

**if (type == typeof(DateTime)) return DateTime.MinValue;**

**return null;**

**}**

**}**

**3. Data Validation and Quality Assurance**

**csharp**

**public class FormDataValidator**

**{**

**public ValidationResult ValidateFormData(EmployeeFormData formData)**

**{**

**var result = new ValidationResult();**

***// Required field validation***

**ValidateRequiredFields(formData, result);**

***// Format validation***

**ValidateFieldFormats(formData, result);**

***// Business rule validation***

**ValidateBusinessRules(formData, result);**

**return result;**

**}**

**private void ValidateRequiredFields(EmployeeFormData formData, ValidationResult result)**

**{**

**if (string.IsNullOrWhiteSpace(formData.FirstName))**

**result.Errors.Add("First Name is required");**

**if (string.IsNullOrWhiteSpace(formData.LastName))**

**result.Errors.Add("Last Name is required");**

**if (string.IsNullOrWhiteSpace(formData.Email))**

**result.Errors.Add("Email is required");**

**if (string.IsNullOrWhiteSpace(formData.Department))**

**result.Errors.Add("Department is required");**

**}**

**private void ValidateFieldFormats(EmployeeFormData formData, result)**

**{**

***// Email validation***

**if (!string.IsNullOrEmpty(formData.Email) && !IsValidEmail(formData.Email))**

**result.Errors.Add("Invalid email format");**

***// Phone validation***

**if (!string.IsNullOrEmpty(formData.PhoneNumber) && !IsValidPhone(formData.PhoneNumber))**

**result.Errors.Add("Invalid phone number format");**

***// Employee ID validation***

**if (!string.IsNullOrEmpty(formData.EmployeeId) && !IsValidEmployeeId(formData.EmployeeId))**

**result.Errors.Add("Invalid Employee ID format (should be EMP######)");**

**}**

**private void ValidateBusinessRules(EmployeeFormData formData, ValidationResult result)**

**{**

***// Salary validation***

**if (formData.Salary < 0)**

**result.Errors.Add("Salary cannot be negative");**

**if (formData.Salary > 1000000)**

**result.Warnings.Add("Salary seems unusually high");**

***// Date validation***

**if (!string.IsNullOrEmpty(formData.StartDate))**

**{**

**if (DateTime.TryParse(formData.StartDate, out var startDate))**

**{**

**if (startDate > DateTime.Now)**

**result.Errors.Add("Start date cannot be in the future");**

**if (startDate < DateTime.Now.AddYears(-50))**

**result.Warnings.Add("Start date seems unusually old");**

**}**

**}**

**}**

**private bool IsValidEmail(string email)**

**{**

**try**

**{**

**var addr = new System.Net.Mail.MailAddress(email);**

**return addr.Address == email;**

**}**

**catch**

**{**

**return false;**

**}**

**}**

**private bool IsValidPhone(string phone)**

**{**

**var cleanPhone = System.Text.RegularExpressions.Regex.Replace(phone, @"[^\d]", "");**

**return cleanPhone.Length == 10;**

**}**

**private bool IsValidEmployeeId(string employeeId)**

**{**

**return System.Text.RegularExpressions.Regex.IsMatch(employeeId, @"^EMP\d{6}$");**

**}**

**}**

**public class ValidationResult**

**{**

**public List<string> Errors { get; set; } = new List<string>();**

**public List<string> Warnings { get; set; } = new List<string>();**

**public bool IsValid => !Errors.Any();**

**public override string ToString()**

**{**

**var messages = new List<string>();**

**if (Errors.Any())**

**messages.Add($"Errors: {string.Join(", ", Errors)}");**

**if (Warnings.Any())**

**messages.Add($"Warnings: {string.Join(", ", Warnings)}");**

**return string.Join(" | ", messages);**

**}**

**}**

**4. Performance Optimization**

**csharp**

**public class OptimizedFormDataRetrieval**

**{**

**private readonly IWebDriver \_driver;**

**private readonly Dictionary<string, IWebElement> \_cachedElements;**

**private DateTime \_lastCacheUpdate;**

**private readonly TimeSpan \_cacheTimeout = TimeSpan.FromMinutes(1);**

**public OptimizedFormDataRetrieval(IWebDriver driver)**

**{**

**\_driver = driver;**

**\_cachedElements = new Dictionary<string, IWebElement>();**

**}**

**public EmployeeFormData GetPageDataOptimized()**

**{**

***// Batch element finding to reduce DOM queries***

**var elementMap = FindAllElementsAtOnce();**

**return new EmployeeFormData**

**{**

**FirstName = GetCachedTextValue("firstName", elementMap),**

**LastName = GetCachedTextValue("lastName", elementMap),**

**Email = GetCachedTextValue("email", elementMap),**

**Department = GetCachedDropdownValue("department", elementMap),**

**IsActive = GetCachedCheckboxValue("isActive", elementMap),**

**RetrievedAt = DateTime.Now,**

**PageUrl = \_driver.Url**

**};**

**}**

**private Dictionary<string, IWebElement> FindAllElementsAtOnce()**

**{**

**var elements = new Dictionary<string, IWebElement>();**

**var selectors = new Dictionary<string, By>**

**{**

**["firstName"] = By.Id("firstName"),**

**["lastName"] = By.Id("lastName"),**

**["email"] = By.Id("email"),**

**["department"] = By.Id("department"),**

**["isActive"] = By.Id("isActive")**

**};**

***// Find all elements in parallel***

**Parallel.ForEach(selectors, kvp =>**

**{**

**try**

**{**

**var element = \_driver.FindElement(kvp.Value);**

**lock (elements)**

**{**

**elements[kvp.Key] = element;**

**}**

**}**

**catch (NoSuchElementException)**

**{**

***// Element not found, continue***

**}**

**});**

**return elements;**

**}**

**private string GetCachedTextValue(string elementKey, Dictionary<string, IWebElement> elementMap)**

**{**

**if (elementMap.TryGetValue(elementKey, out var element))**

**{**

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

**}**

**return "";**

**}**

**private string GetCachedDropdownValue(string elementKey, Dictionary<string, IWebElement> elementMap)**

**{**

**if (elementMap.TryGetValue(elementKey, out var element))**

**{**

**var select = new SelectElement(element);**

**return select.SelectedOption?.Text ?? "";**

**}**

**return "";**

**}**

**private bool GetCachedCheckboxValue(string elementKey, Dictionary<string, IWebElement> elementMap)**

**{**

**if (elementMap.TryGetValue(elementKey, out var element))**

**{**

**return element.Selected;**

**}**

**return false;**

**}**

**}**

**Troubleshooting**

**Common Issues and Solutions**

**1. Element Not Found Exceptions**

**Problem: GetPageData throws NoSuchElementException**

**csharp**

***// Solution: Implement defensive element finding***

**private string GetTextValueSafely(By locator)**

**{**

**try**

**{**

**var wait = new WebDriverWait(\_driver, TimeSpan.FromSeconds(5));**

**var element = wait.Until(d => d.FindElement(locator));**

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

**}**

**catch (WebDriverTimeoutException)**

**{**

**Console.WriteLine($"Element not found: {locator}");**

**return "";**

**}**

**catch (NoSuchElementException)**

**{**

**Console.WriteLine($"Element not found: {locator}");**

**return "";**

**}**

**}**

**2. Stale Element Reference**

**Problem: Elements become stale after page interactions**

**csharp**

***// Solution: Re-find elements when stale***

**private IWebElement FindElementWithRetry(By locator, int maxRetries = 3)**

**{**

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

**{**

**try**

**{**

**return \_driver.FindElement(locator);**

**}**

**catch (StaleElementReferenceException)**

**{**

**if (i == maxRetries - 1) throw;**

**Thread.Sleep(500);**

**}**

**}**

**throw new Exception($"Could not find element after {maxRetries} retries");**

**}**

**3. Timing Issues with Dynamic Content**

**Problem: Form fields are populated after page load**

**csharp**

***// Solution: Wait for specific conditions***

**public EmployeeFormData WaitForFormDataToLoad()**

**{**

**var wait = new WebDriverWait(\_driver, TimeSpan.FromSeconds(10));**

***// Wait for form to be fully loaded***

**wait.Until(d => d.FindElement(By.Id("firstName")).GetAttribute("value") != "");**

**return GetPageData();**

**}**

**4. Iframe and Frame Issues**

**Problem: Form is inside an iframe**

**csharp**

***// Solution: Switch to frame before data extraction***

**public EmployeeFormData GetDataFromFrame(string frameId)**

**{**

**\_driver.SwitchTo().Frame(frameId);**

**try**

**{**

**return GetPageData();**

**}**

**finally**

**{**

**\_driver.SwitchTo().DefaultContent();**

**}**

**}**

**Debug Utilities**

**csharp**

**public class FormDataDebugger**

**{**

**private readonly IWebDriver \_driver;**

**public FormDataDebugger(IWebDriver driver)**

**{**

**\_driver = driver;**

**}**

**public void LogFormState()**

**{**

**Console.WriteLine("=== Form State Debug ===");**

**Console.WriteLine($"Current URL: {\_driver.Url}");**

**Console.WriteLine($"Page Title: {\_driver.Title}");**

**var formElements = \_driver.FindElements(By.TagName("input"))**

**.Concat(\_driver.FindElements(By.TagName("select")))**

**.Concat(\_driver.FindElements(By.TagName("textarea")));**

**foreach (var element in formElements)**

**{**

**try**

**{**

**var id = element.GetAttribute("id");**

**var name = element.GetAttribute("name");**

**var type = element.GetAttribute("type");**

**var value = element.GetAttribute("value");**

**var displayed = element.Displayed;**

**var enabled = element.Enabled;**

**Console.WriteLine($"Element - ID: {id}, Name: {name}, Type: {type}, " +**

**$"Value: {value}, Displayed: {displayed}, Enabled: {enabled}");**

**}**

**catch (Exception ex)**

**{**

**Console.WriteLine($"Error inspecting element: {ex.Message}");**

**}**

**}**

**Console.WriteLine("=== End Form State ===");**

**}**

**public void SavePageScreenshot(string fileName)**

**{**

**if (\_driver is ITakesScreenshot screenshotDriver)**

**{**

**var screenshot = screenshotDriver.GetScreenshot();**

**screenshot.SaveAsFile(fileName, ScreenshotImageFormat.Png);**

**Console.WriteLine($"Screenshot saved: {fileName}");**

**}**

**}**

**public void SavePageSource(string fileName)**

**{**

**File.WriteAllText(fileName, \_driver.PageSource);**

**Console.WriteLine($"Page source saved: {fileName}");**

**}**

**}**

**Performance Monitoring**

**csharp**

**public class FormDataPerformanceMonitor**

**{**

**public class PerformanceMetrics**

**{**

**public TimeSpan DataRetrievalTime { get; set; }**

**public int ElementsFound { get; set; }**

**public int ElementsNotFound { get; set; }**

**public List<string> SlowElements { get; set; } = new List<string>();**

**}**

**public static PerformanceMetrics MeasureFormDataRetrieval(Func<EmployeeFormData> retrievalFunction)**

**{**

**var metrics = new PerformanceMetrics();**

**var stopwatch = Stopwatch.StartNew();**

**var result = retrievalFunction();**

**stopwatch.Stop();**

**metrics.DataRetrievalTime = stopwatch.Elapsed;**

**Console.WriteLine($"Form data retrieval took: {metrics.DataRetrievalTime.TotalMilliseconds}ms");**

**return metrics;**

**}**

**}**

This completes the comprehensive Form Data Retrieval Documentation. The guide covers all aspects from basic data model creation through advanced troubleshooting scenarios, providing a complete reference for implementing robust form data retrieval in automation frameworks.