TIEMPOS DE CASOS Y ESCENARIOS

@trasferAFP  
Feature: Cambio de fondos de pensiones AFP  
 Como usuario de un fondo de pensión AFP  
 Quiero realizar el cambio de fondo de pensión a AFP Integra  
 Para poder obtener mejores beneficios  
  
 @transferAFP\_esc01 @happy  
 Scenario Outline: "<Caso de Prueba>" El usuario valida que el botón "<Posición>" redireccione a la Solicitud de Traspaso a AFP Integra  
 Given que el usuario está en la página principal  
 And el usuario acepta el cookis  
 When el usuario hace clic en el botón de Hazte cliente  
 And el usuario hace clic en el botón Cámbiate Ya  
 And el usuario hace clic en el botón Cámbiarme aquí "<Posición>"  
 Then el usuario valida que ingreso al formulario de solicitud de traspaso  
  
 Examples:  
 | Caso de Prueba | Posición |  
 | CP01 | De la Cabecera |  
 | CP02 | Del Banner |  
 | CP03 | Del Carrusel vertical |  
 | CP04 | Del Contenedor Azul |  
 | CP05 | Del Pie de página |

STEP DEFINITIONS

package com.sura.web.stepdefinitions;  
  
import io.cucumber.java.en.And;  
import io.cucumber.java.en.Then;  
import io.cucumber.java.en.When;  
import questions.ElementVisible;  
import tasks.common.ClickElement;  
import ui.HomePage;  
import ui.TransferPage;  
  
import static org.hamcrest.Matchers.*is*;  
import static net.serenitybdd.screenplay.GivenWhenThen.*seeThat*;  
import static net.serenitybdd.screenplay.actors.OnStage.*theActorInTheSpotlight*;  
  
import net.serenitybdd.screenplay.abilities.BrowseTheWeb;  
import org.openqa.selenium.WebDriver;  
  
  
public class TransferAFPStepDefinitions {  
  
 @When("el usuario hace clic en el botón de Hazte cliente")  
 public void elUsuarioHaceClicEnElBotónDeHazteCliente() {  
 *theActorInTheSpotlight*().attemptsTo(ClickElement.*on*(HomePage.*BUTTON\_HAZTE\_CLIENTE*, true));  
 }  
  
  
 @And("el usuario hace clic en el botón Cámbiate Ya")  
 public void elUsuarioHaceClicEnElBotónCámbiateYa() {  
 *theActorInTheSpotlight*().attemptsTo(ClickElement.*on*(HomePage.*BUTTON\_CAMBIATE\_YA*, true));  
  
 }  
  
 @And("el usuario hace clic en el botón Cámbiarme aquí {string}")  
 public void elUsuarioHaceClicEnElBotónCámbiarmeAquí(String Posicion) {  
 // Obtener el WebDriver asociado al actor  
 WebDriver driver = BrowseTheWeb.*as*(*theActorInTheSpotlight*()).getDriver();  
  
 // Guardar el identificador de la ventana actual  
 String currentWindow = driver.getWindowHandle();  
  
 // Hacer clic en el botón según la posición  
 switch (Posicion.toLowerCase()) {  
 case "de la cabecera":  
 *theActorInTheSpotlight*().attemptsTo(ClickElement.*on*(TransferPage.*CAMBIARMEAQUI\_CABECERA*, true));  
 break;  
 case "del banner":  
 *theActorInTheSpotlight*().attemptsTo(ClickElement.*on*(TransferPage.*CAMBIARMEAQUI\_BANNER*, true));  
 break;  
 case "del carrusel vertical":  
 *theActorInTheSpotlight*().attemptsTo(ClickElement.*on*(TransferPage.*CAMBIARMEAQUI\_CARRUSEL*, true));  
 break;  
 case "del contenedor azul":  
 *theActorInTheSpotlight*().attemptsTo(ClickElement.*on*(TransferPage.*CAMBIARMEAQUI\_CONTENEDOR*, true));  
 break;  
 case "del pie de página":  
 *theActorInTheSpotlight*().attemptsTo(ClickElement.*on*(TransferPage.*CAMBIARMEAQUI\_PIE*, true));  
 break;  
 default:  
 throw new IllegalArgumentException("No se reconoce el botón: " + Posicion);  
 }  
  
 // Cambiar a la nueva ventana y cerrar la anterior  
 driver.getWindowHandles()  
 .stream()  
 .filter(window -> !window.equals(currentWindow)) // Filtrar para obtener la nueva ventana  
 .findFirst()  
 .ifPresent(newWindow -> {  
 driver.switchTo().window(currentWindow).close(); // Cerrar la ventana actual  
 driver.switchTo().window(newWindow); // Cambiar a la nueva ventana  
 });  
  
 }  
  
 @Then("el usuario valida que ingreso al formulario de solicitud de traspaso")  
 public void elUsuarioValidaQueIngresoAlFormularioDeSolicitudDeTraspaso() {  
 *theActorInTheSpotlight*().should(*seeThat*(ElementVisible.*isVisible*(TransferPage.*VALIDAR\_SOLICITUD*),*is*(true)));  
 }  
}

USER INTERFACE

package ui;  
  
import net.serenitybdd.screenplay.targets.Target;  
  
public class TransferPage {  
  
 public static final Target *CAMBIARMEAQUI\_CABECERA* = Target.*the*("clic en botón cambiarme aquí cabecera")  
 .locatedBy("//a[@id='cambiarme\_aqui\_header']");  
  
 public static final Target *CAMBIARMEAQUI\_BANNER* = Target.*the*("clic en botón cambiarme aquí banner")  
 .locatedBy("//a[@id='cambiarme\_aqui\_segundo']");  
  
 public static final Target *CAMBIARMEAQUI\_CARRUSEL* = Target.*the*("clic en botón cambiarme aquí carrusel")  
 .locatedBy("//a[@id='cambiarme\_aqui\_tercero']");  
  
 public static final Target *CAMBIARMEAQUI\_CONTENEDOR* = Target.*the*("clic en botón cambiarme aquí contenedor")  
 .locatedBy("(//div[text()='Cambiarme aquí'])[4]/parent::a");  
  
 public static final Target *CAMBIARMEAQUI\_PIE* = Target.*the*("clic en botón cambiarme aquí pie de página")  
 .locatedBy("(//div[text()='Cambiarme aquí'])[6]/parent::a");  
  
 public static final Target *VALIDAR\_SOLICITUD* = Target.*the*("valida pantalla inicio de solicitud")  
 .locatedBy("(//p[contains(text(), '¡Empecemos con tu cambio a AFP Integra!')])[1]");  
  
  
}

QUESTIONS

package questions;  
  
import net.serenitybdd.screenplay.Actor;  
import net.serenitybdd.screenplay.Question;  
import net.serenitybdd.screenplay.actions.Scroll;  
import net.serenitybdd.screenplay.matchers.WebElementStateMatchers;  
import net.serenitybdd.screenplay.targets.Target;  
import net.serenitybdd.screenplay.waits.WaitUntil;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
public class ElementVisible implements Question<Boolean> {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(ElementVisible.class);  
 private final Target target;  
 private final String[] values;  
 private static final int *TIMEOUT\_IN\_SECONDS* = 15;  
  
 public ElementVisible(Target target, String... values) {  
 this.target = target;  
 this.values = values;  
 }  
  
 @Override  
 public Boolean answeredBy(Actor actor) {  
 try {  
 // Determinar el Target final basado en si se proporcionaron valores o no  
 Target finalTarget = (values == null || values.length == 0) ? target : target.of(values);  
  
 actor.attemptsTo(  
 // Esperar hasta que el elemento sea visible  
 WaitUntil.*the*(finalTarget, WebElementStateMatchers.*isVisible*()).forNoMoreThan(*TIMEOUT\_IN\_SECONDS*).seconds(),  
 // Desplazarse al elemento  
 Scroll.*to*(finalTarget)  
 );  
 return finalTarget.resolveFor(actor).isVisible();  
 } catch (Exception e) {  
 *LOGGER*.error("Error while waiting for element to be visible: {}", e.getMessage());  
 return false;  
 }  
 }  
  
 public static ElementVisible isVisible(Target target, String... values) {  
 return new ElementVisible(target, values);  
 }  
}

TASK

package tasks.common;  
  
import net.serenitybdd.screenplay.Actor;  
import net.serenitybdd.screenplay.Task;  
import net.serenitybdd.screenplay.actions.Click;  
import net.serenitybdd.screenplay.actions.Scroll;  
import net.serenitybdd.screenplay.targets.Target;  
import net.serenitybdd.screenplay.waits.WaitUntil;  
  
import static net.serenitybdd.screenplay.matchers.WebElementStateMatchers.*isClickable*;  
import static net.serenitybdd.screenplay.matchers.WebElementStateMatchers.*isVisible*;  
  
public class ClickElement implements Task {  
  
 private final Target target;  
 private final boolean shouldScroll;  
 private final String[] values;  
  
 // Constructor que acepta un Target, un indicador de desplazamiento y un número variable de valores  
 public ClickElement(Target target, boolean shouldScroll, String... values) {  
 this.target = target;  
 this.shouldScroll = shouldScroll;  
 this.values = values;  
 }  
  
 @Override  
 public <T extends Actor> void performAs(T actor) {  
 // Determinar el Target final basado en si se proporcionaron valores o no  
 Target finalTarget = (values == null || values.length == 0) ? target : target.of(values);  
  
 // Desplazarse al elemento si shouldScroll es verdadero  
 if (shouldScroll) {  
 actor.attemptsTo(Scroll.*to*(finalTarget));  
 }  
  
 // Esperar hasta que el elemento sea visible y clickeable  
 actor.attemptsTo(  
 WaitUntil.*the*(finalTarget, *isVisible*()).forNoMoreThan(15).seconds(),  
 WaitUntil.*the*(finalTarget, *isClickable*()).forNoMoreThan(15).seconds(),  
 // Hacer clic en el elemento  
 Click.*on*(finalTarget)  
 );  
 }  
  
 // Mé*todo estático para facilitar la creación de la tarea*  
public static ClickElement on(Target target, boolean shouldScroll, String... values) {  
 return new ClickElement(target, shouldScroll, values);  
 }  
}  
  
//package tasks.common;  
//  
//import net.serenitybdd.screenplay.Actor;  
//import net.serenitybdd.screenplay.Task;  
//import net.serenitybdd.screenplay.actions.Click;  
//import net.serenitybdd.screenplay.actions.Scroll;  
//import net.serenitybdd.screenplay.targets.Target;  
//import net.serenitybdd.screenplay.waits.WaitUntil;  
//  
//import static net.serenitybdd.screenplay.matchers.WebElementStateMatchers.isVisible;  
//  
//public class ClickElement implements Task {  
//  
// private final Target target;  
// private final String[] values;  
//  
// // Constructor que acepta un Target y un número variable de valores  
// public ClickElement(Target target, String... values) {  
// this.target = target;  
// this.values = values;  
// }  
//  
// @Override  
// public <T extends Actor> void performAs(T actor) {  
// // Determinar el Target final basado en si se proporcionaron valores o no  
// Target finalTarget = (values == null || values.length == 0) ? target : target.of(values);  
//  
// actor.attemptsTo(  
// // Desplazarse al elemento  
// Scroll.to(finalTarget),  
// // Esperar hasta que el elemento sea visible  
// WaitUntil.the(finalTarget, isVisible()).forNoMoreThan(15).seconds(),  
// // Hacer clic en el elemento  
// Click.on(finalTarget)  
// );  
// }  
//  
// // Mé*todo estático para facilitar la creación de la tarea*  
// public static ClickElement on(Target target, String... values) {  
// return new ClickElement(target, values);  
// }  
//}

OPCIONAL

package tasks.common;  
  
import net.serenitybdd.screenplay.Actor;  
import net.serenitybdd.screenplay.Task;  
import net.serenitybdd.screenplay.actions.Enter;  
import net.serenitybdd.screenplay.actions.Scroll;  
import net.serenitybdd.screenplay.targets.Target;  
import net.serenitybdd.screenplay.waits.WaitUntil;  
  
import static net.serenitybdd.screenplay.matchers.WebElementStateMatchers.*isVisible*;  
  
public class EnterText implements Task {  
  
 private final Target target;  
 private final String value;  
  
 public EnterText(Target target, String value) {  
 this.target = target;  
 this.value = value;  
 }  
  
 @Override  
 public <T extends Actor> void performAs(T actor) {  
 actor.attemptsTo(  
 // Desplazarse al elemento  
 Scroll.*to*(target),  
 // Esperar hasta que el elemento sea visible  
 WaitUntil.*the*(target, *isVisible*()).forNoMoreThan(15).seconds(),  
 // Ingresar el valor en el elemento  
 Enter.*theValue*(value).into(target)  
 );  
 }  
  
 // Mé*todo estático para facilitar la creación de la tarea*  
public static EnterText into(Target target, String value) {  
 return new EnterText(target, value);  
 }  
}

SERENITY

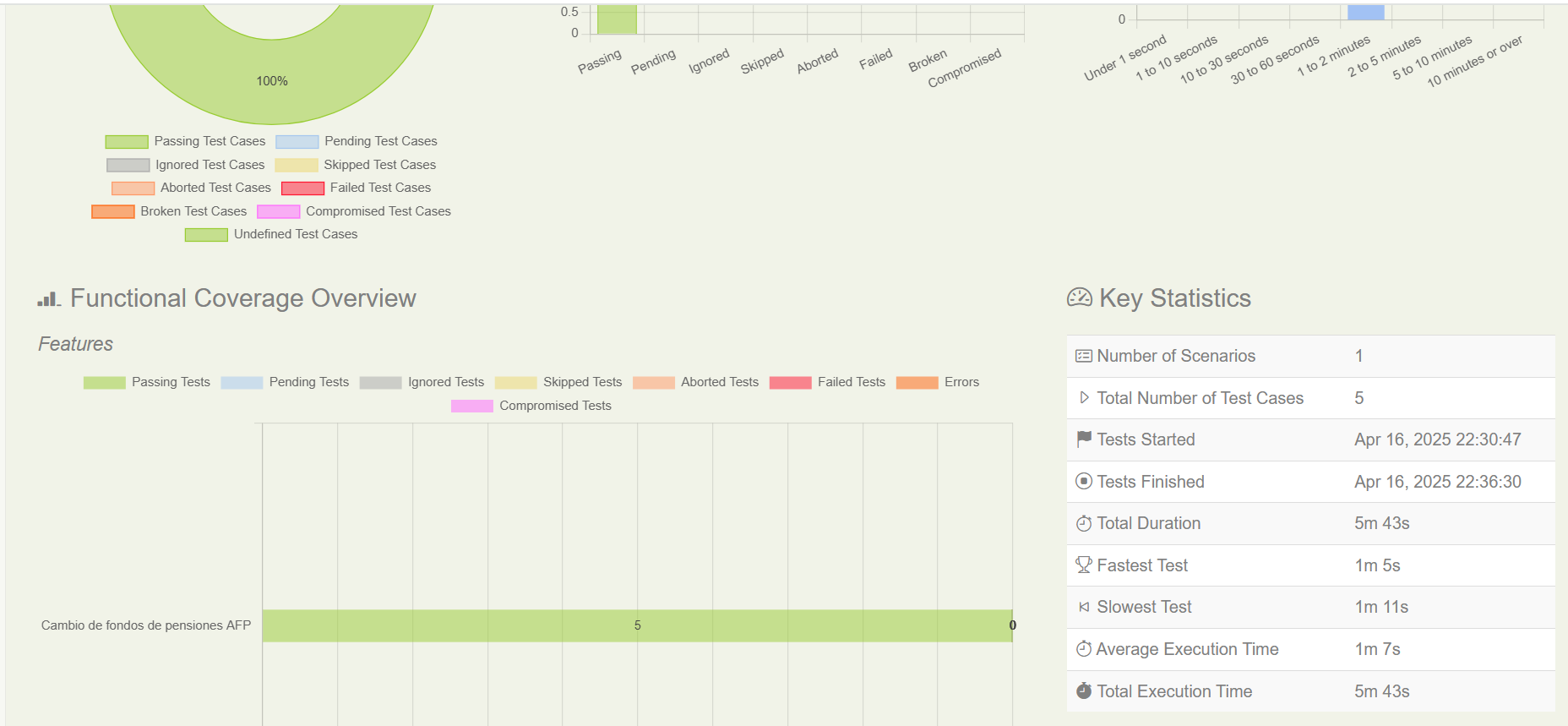
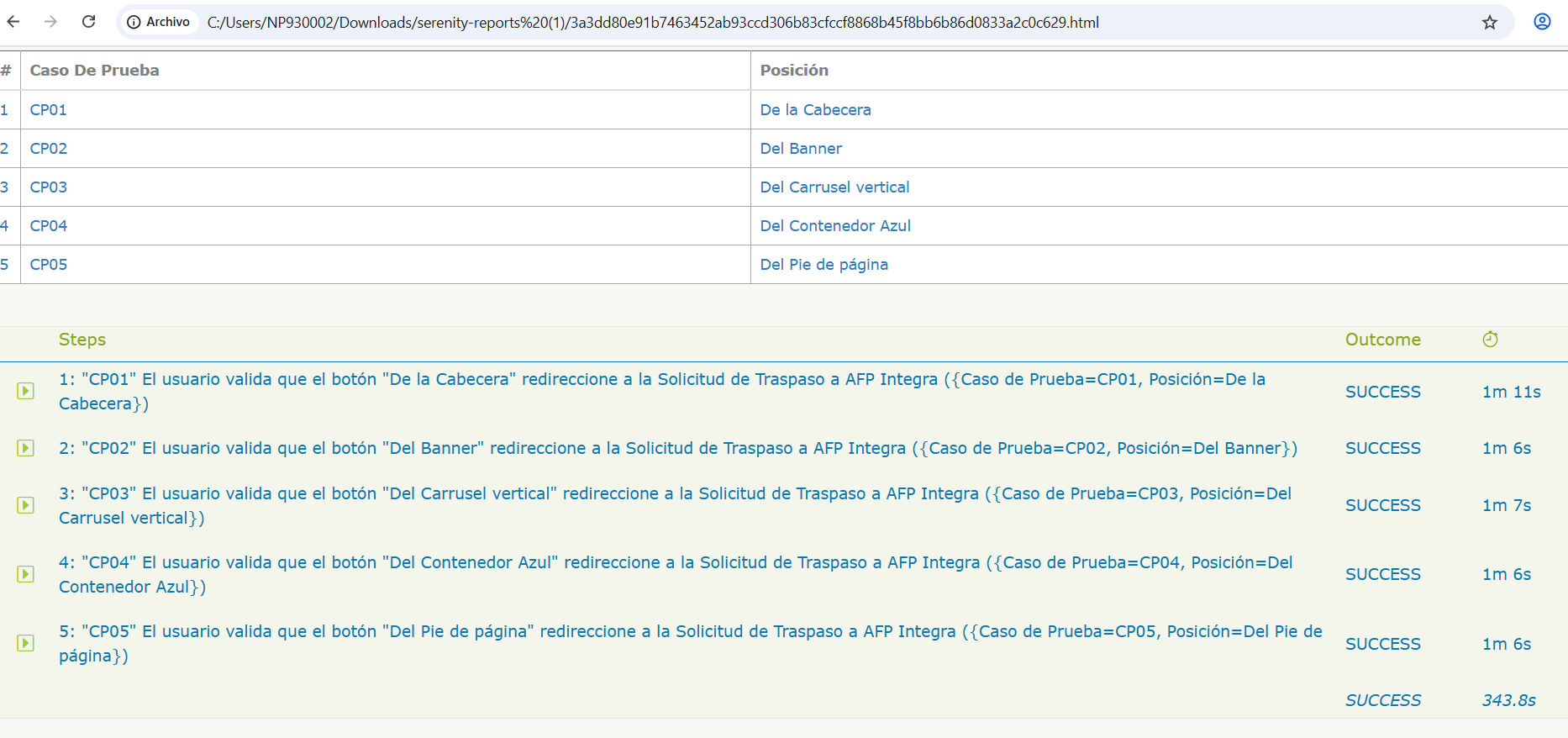
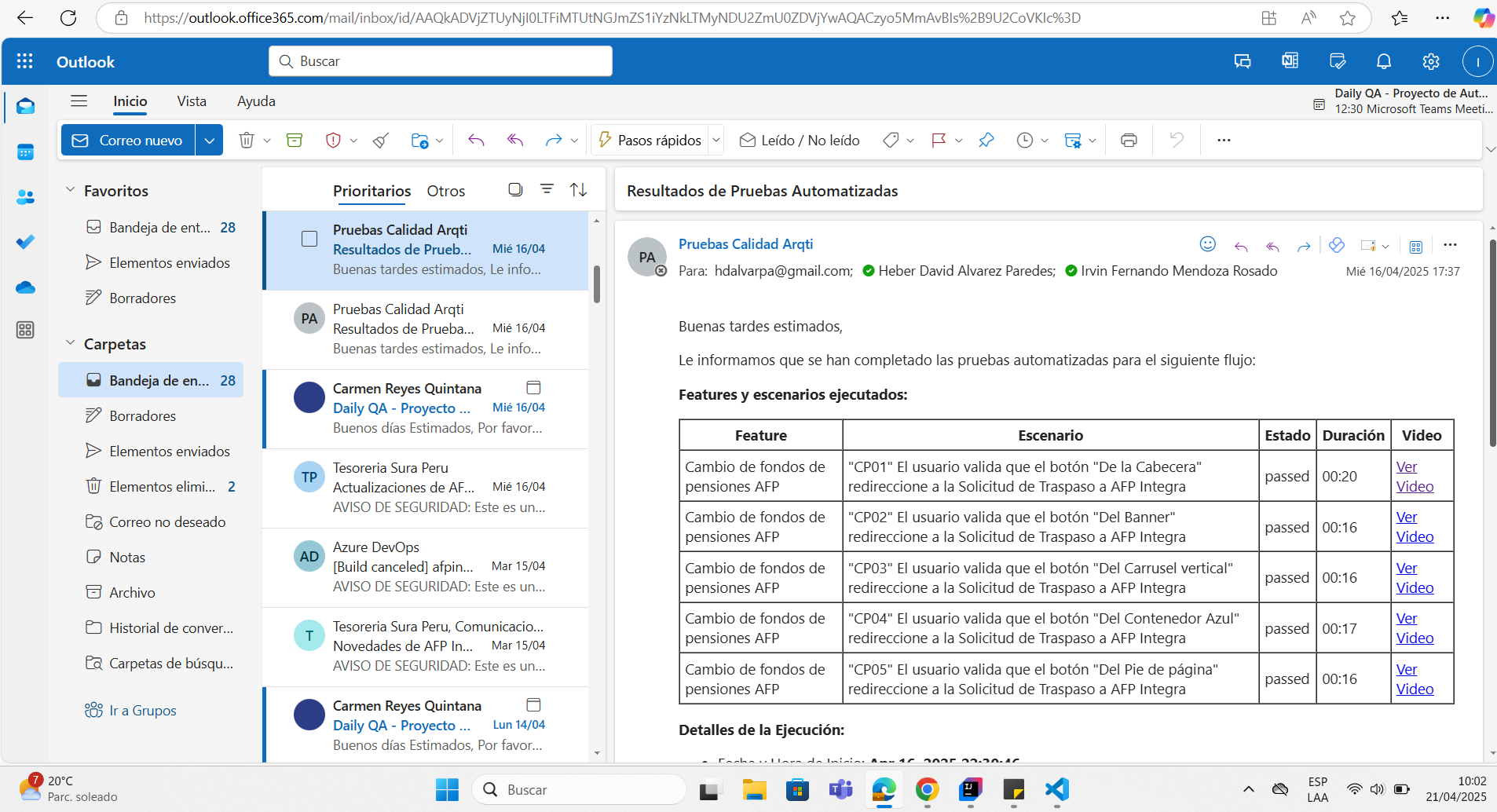
serenity {  
 encoding = "UTF-8"  
 compress.filenames = true  
 take.screenshots = FOR\_FAILURES *# Cambiar a FOR\_FAILURES para capturar solo en fallos*  
report.include.step.details = false *# Excluir detalles adicionales de pasos*  
report.include.hooks = false *# Excluir tiempos de hooks (@Before y @After)*  
report.show.manual.tests = false *# No mostrar pruebas manuales*  
report.show.released.tests = false *# No mostrar pruebas liberadas*  
report.generate.manual.tests = false *# No generar evidencias de pruebas manuales*  
report.generate.released.tests = false *# No generar evidencias de pruebas liberadas*  
 arn {  
 deviceFarm = "arn:aws:devicefarm:us-west-2:740893539825:testgrid-project:55a2dd70-1fe9-47eb-aa64-e16ffd3371e4"  
 }  
}  
  
 environments {  
 default{  
 login.url = "<https://test.afpintegra.pe/iniciar-sesion>"  
 principal.url = "<https://test.afpintegra.pe/>"  
 outlook.url = "<https://go.microsoft.com/fwlink/p/?LinkID=2125442&amp;clcid=0x409&amp;culture=en-us&amp;country=us>"  
 }  
 dev {  
 login.url = "<https://test.afpintegra.pe/iniciar-sesion>"  
 principal.url = "<https://test.afpintegra.pe/>"  
 outlook.url = "<https://go.microsoft.com/fwlink/p/?LinkID=2125442&amp;clcid=0x409&amp;culture=en-us&amp;country=us>"  
 }  
}

@Given("que el usuario está en la página principal")  
public void queElUsuarioEstáEnLaPáginaPrincipal() {  
 *theActorCalled*("user").wasAbleTo(PrincipalUrl.*CambioAFP*());  
}

@And("el usuario acepta el cookis")  
public void elUsuarioAceptaElCookis() {  
 *theActorInTheSpotlight*().attemptsTo(ButtonCookis.*close*());  
}

package tasks.changeFund;  
  
import net.serenitybdd.screenplay.Actor;  
import net.serenitybdd.screenplay.Task;  
import net.serenitybdd.screenplay.actions.Click;  
import net.serenitybdd.screenplay.actions.Scroll;  
import net.serenitybdd.screenplay.conditions.Check;  
import net.serenitybdd.screenplay.waits.WaitUntil;  
import tasks.dashboard.CloseModalPresent;  
import ui.ChangeFundPage;  
import ui.CookiesPage;  
import ui.DashboardPage;  
  
import static net.serenitybdd.screenplay.matchers.WebElementStateMatchers.*isVisible*;  
  
public class ButtonCookis implements Task {  
 @Override  
 public <T extends Actor> void performAs(T actor) {  
 actor.attemptsTo(  
 // Verifica si el botón de aceptar cookies es visible  
 Check.*whether*(CookiesPage.*COOKIES\_ACCEPT*.resolveFor(actor).isVisible())  
 .andIfSo(  
 Scroll.*to*(CookiesPage.*COOKIES\_ACCEPT*),  
 Click.*on*(CookiesPage.*COOKIES\_ACCEPT*)  
 )  
 );  
 }  
  
 // Metodo estático para facilitar la creación de la tarea  
 public static ButtonCookis close() {  
 return new ButtonCookis();  
 }  
}

public class CookiesPage {  
 public static final Target *COOKIES\_ACCEPT* = Target.*the*("Accept Cookies")  
 .located(By.*id*("btn\_accept\_all\_cookies"));





Falto

package tasks.login;  
  
import net.serenitybdd.annotations.Step;  
import net.serenitybdd.model.environment.ConfiguredEnvironment;  
import net.serenitybdd.model.environment.EnvironmentSpecificConfiguration;  
import net.serenitybdd.screenplay.Actor;  
import net.serenitybdd.screenplay.Task;  
import net.serenitybdd.screenplay.actions.Open;  
import net.thucydides.model.util.EnvironmentVariables;  
  
import static net.serenitybdd.screenplay.Tasks.*instrumented*;  
  
public class PrincipalUrl implements Task {  
  
 private final String PrincipalUrl;  
 private final EnvironmentVariables environmentVariables;  
  
 public PrincipalUrl(String PrincipalUrl) {  
 this.PrincipalUrl = PrincipalUrl;  
 this.environmentVariables = ConfiguredEnvironment.*getEnvironmentVariables*();  
 }  
  
  
 public static Task CambioAFP() {  
 String PrincipalUrl = "principal.url";  
 return *instrumented*(PrincipalUrl.class, PrincipalUrl);  
 }  
  
 @Override  
 @Step("{0} Inicia la página #url")  
 public <T extends Actor> void performAs(T actor) {  
  
 String targetUrl = EnvironmentSpecificConfiguration.*from*(environmentVariables).getProperty(PrincipalUrl);  
  
 if (targetUrl == null) {  
 throw new IllegalArgumentException("URL not found in environment variables: " + PrincipalUrl);  
 } else {  
 actor.attemptsTo(Open.*url*(targetUrl));  
 }  
 }  
}

@And("el usuario acepta el cookis")  
public void elUsuarioAceptaElCookis() {  
 *theActorInTheSpotlight*().attemptsTo(WaitFor.*seconds*(2)); //tiempo explicito  
 *theActorInTheSpotlight*().attemptsTo(ClickElement.*on*(CookiesPage.*COOKIES\_ACCEPT*, true));  
}

package ui;  
  
import net.serenitybdd.screenplay.targets.Target;  
  
public class HomePage {  
  
 public static final Target *BUTTON\_HAZTE\_CLIENTE* = Target.*the*("clic en botón hazte cliente")  
 .locatedBy("//button[@id='home\_\_show\_modal\_\_become\_a\_customer']");  
  
 public static final Target *BUTTON\_CAMBIATE\_YA* = Target.*the*("clic en botón cámbiate ya")  
 .locatedBy("//a[@id='home\_\_modal\_\_change\_to\_integra']");  
  
  
}

package tasks.common;  
  
import net.serenitybdd.screenplay.Actor;  
import net.serenitybdd.screenplay.Task;  
import net.serenitybdd.screenplay.targets.Target;  
import org.openqa.selenium.JavascriptExecutor;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.WebElement;  
  
import static net.serenitybdd.screenplay.Tasks.*instrumented*;  
  
public class ScrollTo implements Task {  
 private final Target target;  
 public ScrollTo(Target target) {  
 this.target = target;  
 }  
  
 public static ScrollTo element(Target target) {  
 return *instrumented*(ScrollTo.class, target);  
 }  
  
 @Override  
 public <T extends Actor> void performAs(T actor) {  
 WebDriver driver = net.serenitybdd.screenplay.abilities.BrowseTheWeb.*as*(actor).getDriver();  
 WebElement element = target.resolveFor(actor);  
 // Usar JavaScript para hacer scroll al elemento  
 ((JavascriptExecutor) driver).executeScript("arguments[0].scrollIntoView(true);", element);  
 }  
}

package tasks.common;  
  
import net.serenitybdd.screenplay.Actor;  
import net.serenitybdd.screenplay.Task;  
  
import static net.serenitybdd.screenplay.Tasks.*instrumented*;  
  
public class WaitFor implements Task {  
 private final int seconds;  
  
 public WaitFor(int seconds) {  
 this.seconds = seconds;  
 }  
  
 public static WaitFor seconds(int seconds) {  
 return *instrumented*(WaitFor.class, seconds);  
 }  
  
 @Override  
 public <T extends Actor> void performAs(T actor) {  
 try {  
 Thread.*sleep*(seconds \* 1000L); // Convertir segundos a milisegundos  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt(); // Restaurar el estado de interrupción  
 throw new RuntimeException("La espera fue interrumpida", e);  
 }  
 }  
}

package questions;  
  
import net.serenitybdd.screenplay.Question;  
import ui.ChangeFundPage;  
  
public class ChangeFundQuestions {  
  
 public static Question<String> ValidateMessage(String expectedText) {  
 return actor -> ChangeFundPage.*H5\_LABEL*.of(expectedText).resolveFor(actor).waitUntilVisible().getText().trim();  
 }  
  
 public static Question<String> ValidateMessageChangeFund(String expectedText) {  
 return actor -> ChangeFundPage.*LBL\_MESSAGE\_CHANGE\_FUND*.of(expectedText).resolveFor(actor).waitUntilVisible().getText().trim();  
 }  
}