CISC 327 Group 34 - CH

Assignment 6

December 3, 2024

Hayden Jenkins 20344883
Ryan Van Drunen 20331633
Madison MacNeil 20285877

Github Username – Name | Student Number

ryanvandrunen - Ryan Van Drunen | 20331633
hjenkins04/xonixdev - Hayden Jenkins | 20344883
madisonmacneil - Madison MacNeil | *20285877*

Task	Contributor
Dashboard End-to-End Tests	Ryan Van Drunen
Search Results End-to-End Tests	Madison MacNeil
Homepage End-to-End Tests	Madison MacNeil
SeatBooking End-to-End Test	Hayden Jenkins
Login End-to-End Test	Hayden Jenkins

How To Run End-To-End Tests

Follow these steps:

1. Navigate to the `roam-end2end-tests` directory. Open a terminal and type `cd roam-end2end-tests`.
2. Create a virtual environment in the `roam-backend` directory. Use the command `python -m venv venv`.
3. Activate the virtual environment. On macOS/Linux, type `source venv/bin/activate`. On Windows, type `.\venv\Scripts\activate`.
4. Confirm the virtual environment is activated. Once activated, your terminal prompt will show `(venv)` at the beginning.
5. Install the required packages listed in `requirements.txt` by typing `pip install -r requirements.txt`.
6. Run the web application by following the steps outlined in the **How to Run the App** section below.
7. Run the End-to-End tests. Use the command `pytest -n auto -v tests`.

8. Alternatively, you can run the tests using `run.py` or `run_pytest`. For manual execution, type `python run.py`. For pytest execution, type `python run_pytest.py`.

How to Run the App

To run the project with both the frontend and backend, follow these steps:

1. Install Docker. Download and install Docker from the official Docker website.
2. Navigate to the repository directory. Open a terminal and ensure you are in the project's root directory.
3. Build the Docker containers by typing `docker compose build`.
4. Start the containers by typing `docker compose up -d`.
5. Verify that both the frontend and backend containers are running by typing `docker ps`.
6. Open your browser and go to `http://localhost:3000` to access the web application.
7. To view the Docker images created during the process, type `docker images`.
8. Follow the instructions in the "How to Run End-to-End Tests" section to run the tests.

Dashboard End-To-End Tests

Step 1: Open and Log In

This ensures that the application home page is loaded and the user is logged in.

```
# Navigate to Profile Page
self.driver.get("http://localhost:3000")
self.logger.debug("Opened the application home page.")

# Ensure user is logged in
self.ensure_logged_in()
self.logger.debug("User is logged in.")
```

Step 2: Profile Dashboard Navigation

Clicking the user avatar to navigate to the dashboard.

```
# Navigate to Dashboard
user_avatar = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="user-avatar"]'))
)
user_avatar.click()
self.logger.debug("Clicked the user avatar.")

dashboard_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="dashboard-button"]'))
)
dashboard_button.click()
self.logger.debug("Clicked the dashboard button.")
```

Step 3: Profile Header and Sidebar Validation

Validate that the Profile Header and Sidebar are visible on the dashboard.

```
# Check for Profile Header
profile_header = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.CSS_SELECTOR, '[data-testid="profile-header"]'))
)
assert profile_header.is_displayed(), "Profile Header not visible"
self.logger.debug("Profile Header is visible.")

# Check for Sidebar
sidebar = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.CSS_SELECTOR, '[data-testid="profile-sidebar"]'))
)
assert sidebar.is_displayed(), "Sidebar not visible"
self.logger.debug("Sidebar is visible.")
```

Step 4: Navigate to Edit Profile

Click the "Edit Profile" button to navigate to the Edit Profile form.

```
# Navigate to Edit Profile
edit_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="edit-profile-button"]'))
)
edit_button.click()
self.logger.debug("Clicked Edit Profile button.")
```

Step 5: Fill and Submit Edit Profile Form

Verify that the form can be filled with new data and submitted successfully.

```python
# Fill Edit Form
first_name_input = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.XPATH, '//input[@placeholder="First Name*"]'))
)
first_name_input.clear()
first_name_input.send_keys("NewFirstName")
self.logger.debug("Entered new first name.")

last_name_input = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.XPATH, '//input[@placeholder="Last Name*"]'))
)
last_name_input.clear()
last_name_input.send_keys("NewLastName")
self.logger.debug("Entered new last name.")

dob_input = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.XPATH, '//input[@placeholder="Date of Birth*"]'))
)
dob_input.clear()
dob_input.send_keys("01/01/2000")
self.logger.debug("Entered new date of birth.")

email_input = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.XPATH, '//input[@placeholder="Email Address*"]'))
)
email_input.clear()
email_input.send_keys("update@email.com")
self.logger.debug("Entered new address.")

phone_input = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.XPATH, '//input[@placeholder="Phone Number*"]'))
)
phone_input.clear()
phone_input.send_keys("1234567890")
self.logger.debug("Entered new phone number.")

address_input = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.XPATH, '//input[@placeholder="Street Address*"]'))
)
address_input.clear()
address_input.send_keys("NewAddress")
self.logger.debug("Entered new address.")
```

```python
province_input = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.XPATH, '//input[@placeholder="Province*"]'))
)
province_input.clear()
province_input.send_keys("NewProvince")
self.logger.debug("Entered new province.")

postal_code_input = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.XPATH, '//input[@placeholder="Postal Code*"]'))
)
postal_code_input.clear()
postal_code_input.send_keys("A1A1A1")
self.logger.debug("Entered new postal code.")

submit_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="submit-button"]'))
)
submit_button.click()
self.logger.debug("Submitted profile form.")
```

Step 6: Handle Success Modal

Ensure the success modal is displayed after form submission, then close it.

```
# Verify Success Modal
success_modal = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.XPATH, "//h2[text()='Account Updated']"))
)
assert success_modal.is_displayed(), "Success modal not displayed."
self.logger.debug("Profile updated successfully.")

# Close Success Modal
back_button = WebDriverWait(self.driver, 100).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="close-button"]'))
)
back_button.click()
self.logger.debug("Closed success modal.")
```

Step 7: Navigate to Purchases

Click the "Purchases" button to navigate to the Purchases section.

```
# Navigate to purchases section
purchases_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="purchases-button"]'))
)
purchases_button.click()
self.logger.debug("Clicked Purchases button.")
```

Step 8: Refund a Trip/Ticket

If a purchase exists, initiate a refund by clicking the cancel icon, and confirm the refund.

```
# Handle Refunds (if any purchases exist)
refund_button = self.driver.find_elements(By.CSS_SELECTOR, '[data-testid="cancel-icon"]')
if refund_button:
    refund_button[0].click()
    self.logger.debug("Initiated refund.")

    # Confirm Refund Dialog
    confirm_button = WebDriverWait(self.driver, 10).until(
        EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="refund-confirm-button"]'))
    )
    confirm_button.click()
    self.logger.debug("Confirmed refund.")
```

Homepage End-To-End Tests

Step 1: Open and Login

This ensures that the application home page is loaded, and the user is logged in.

```
self.driver.get("http://localhost:3000/")
self.logger.debug("Opened the application homepage.")

self.driver.set_window_size(968, 533)
self.logger.debug("Set the browser window size.")

self.ensure_logged_in()
self.logger.debug("User is logged in.")
```

Step 2: Select Arrival and Departure Cities

This step makes selections from the Departure City and Arrival City dropdown buttons. The departure city is chosen to be Vancouver, selecting Vancouver International Airport, and the arrival airport is chosen to be Toronto, selecting Toronto Pearson International Airport.

```
#===SELECT ARRIVAL & DEPARTURE CITIES====

departure_city = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="departure-city"]'))
)
departure_city.click()
self.logger.debug("Clicked Departure City button.")

depart_vancouver = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="airport-item-YVR"]'))
)
depart_vancouver.click()
self.logger.debug("Selected Vancouver Departure.")

departure_city_close = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="departure-city"]'))
)
departure_city_close.click()
self.logger.debug("Closed Departure City button.")

time.sleep(1)

arrival_city = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="arrival-city"]'))
)
arrival_city.click()
self.logger.debug("Clicked Arrival City button.")

arrive_toronto = WebDriverWait(self.driver, 20).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="airport-item-YYZ"]'))
)
arrive_toronto.click()
self.logger.debug("Selected Toronto Arrival.")
```

Step 3: Select Arrival and Departure Dates

This step makes selections from the Departure Date and Arrival Date dropdown calendars. The departure date to be is chosen to be December 21st 2024 and the arrival date is chosen to be December 25th 2024.

```
#===SELECT DEPARTURE AND ARRIVAL DATES====

departure_date_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="departure-date"]'))
)
departure_date_button.click()
self.logger.debug("Clicked Departure Date Button.")

departure_date = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.XPATH, '//button[text()="21"]'))
    )
departure_date.click()
self.logger.debug("Selected Arrival Date: 2024-12-21.")

departure_date_close = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="departure-date"]'))
)
departure_date_close.click()
self.logger.debug("Closed Departure Date Button.")

time.sleep(1)

arrival_date_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="return-date"]'))
)
arrival_date_button.click()
self.logger.debug("Clicked Arrival Date Button.")

arrival_date = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.XPATH, '//button[text()="25"]'))
    )
arrival_date.click()
self.logger.debug("Selected Arrival Date: 2024-12-25.")
```

Step 4: Search for Flights

This step clicks the search button, which routes the user to a new page, provided a departure and arrival cities, as well as departure and arrival dates have been inputted. The traveller and class button does not need to receive input for the search button to route to the next page, it operates with the default of one business class passenger.

```
#==== SEARCH ====

search_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="search-button"]'))
)
search_button.click()
self.logger.debug("Clicked Search Button.")
```

Search Results End-To-End Tests

Step 1: Open and Login

This ensures that the application home page is loaded, and the user is logged in, then navigates to the search result page by first navigating the home page.

```
self.driver.get("http://localhost:3000/")
self.logger.debug("Opened the application homepage.")

self.driver.set_window_size(968, 533)
self.logger.debug("Set the browser window size.")

self.ensure_logged_in()
self.logger.debug("User is logged in.")

self.homepage_navigation()
```

Step 2: Update Departure and Arrival City Values

This step updates the values of departure and arrival cities. *The departure city is chosen to be Paris, selecting Charles de Gaulle International Airport, and the arrival airport is chosen to be Sao Paulo, selecting Sao Paulo International Airport.*

```
#=== UPDATE DEPARTURE + ARRIVAL CITY ===

departure_city = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="departure-city-button"]'))
)
departure_city.click()
self.logger.debug("Clicked Departure City button.")

depart_vancouver = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="airport-item-CDG"]'))
)
depart_vancouver.click()
self.logger.debug("Selected Paris Departure.")

time.sleep(1)

arrival_city = WebDriverWait(self.driver, 20).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="arrival-city-button"]'))
)
arrival_city.click()
self.logger.debug("Clicked Arrival City button.")

time.sleep(1)

arrive_toronto = WebDriverWait(self.driver, 20).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="airport-item-GRU"]'))
)
arrive_toronto.click()
self.logger.debug("Selected Sao Paolo Arrival.")

time.sleep(2)
```

<u>Step 3:</u> Confirm Flight Rendering

This step confirms that a flight provided by Air France renders as a search result item.

```
#===CONFIRM FLIGHT RENDERING===

air_france_flights = self.driver.find_elements(By.XPATH, "//*[contains(text(), 'Air France')]")
assert len(air_france_flights) == 1
self.logger.debug("Air France Flights Render.")
```

<u>Step 4:</u> Apply Airline Filter to Results

This step applies an airline filter to the search results. The Airline filter button is clicked, and LATAM Airlines is chosen as the filter from the dropdown. The search button is pressed to update the search results according to the filters.

```
#===APPLY A FILTER ===

airline_filter = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="filter-button-5"]'))
)
airline_filter.click()
self.logger.debug("Clicked Airline Filter Button.")

filter_selection = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="dropdown-selection-0"]'))
)
filter_selection.click()
self.logger.debug("Airline Filter Selection Made: LATAM Airlines.")

search_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="search-button"]'))
)
search_button.click()
self.logger.debug("Clicked Search Button.")

time.sleep(1)
```

<u>Step 5:</u> Confirm Filtered Flight Rendering

This step confirms that a flight provided by Air France no longer renders in the search results, now that a filter for flights provided by LATAM Airlines has been applied.

```
#===CONFIRM FILTERED RENDERING===

air_france_flights = self.driver.find_elements(By.XPATH, "//*[contains(text(), 'Air France')]")
assert len(air_france_flights) == 0
self.logger.debug("No Air France Flights Render.")
```

Step 6: Apply Max Price Filter to Results

This step applies another filter to the search results. It clicks the Max Price button and selects $200 from the dropdown. The search button is then pressed to apply the new filter to the search results.

```
#===FILTER AGAIN===

max_price_filter = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="filter-button-1"]'))
)
max_price_filter.click()
self.logger.debug("Clicked Price Filter Button.")

filter_selection = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="dropdown-selection-0"]'))
)
filter_selection.click()
self.logger.debug("Max Price Filter Selection Made: $200.")

search_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="search-button"]'))
)
search_button.click()
self.logger.debug("Clicked Search Button.")

time.sleep(1)
```

Step 7: Confirm Filtered Flight Rendering:

This step confirms that no flights render after the application of the Max Price filter, as all flights between these cities by LATAM airlines are priced at greater than $200. The text "No results found for your search criteria" is confirmed to appear on the page.

```
#===CONFIRM FILTERED RENDERING===

no_results = self.driver.find_element(By.XPATH, "//*[contains(text(), 'No results found for your search criteria.')]
assert no_results
self.logger.debug("No Search Results Found.")
```

Step 8: Undo Max Price Filter:

This step undoes the previously applied Max Price filter. The Max Price button is clicked and the reset option in the dropdown is selected, then the search button is pressed to render search results not subject to the $200 max price filter.

```
#=== UNDO PRICE FILTER ===

max_price_filter = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="filter-button-1"]'))
)
max_price_filter.click()
self.logger.debug("Clicked Price Filter Button.")

time.sleep(1)

reset_filter = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="reset-button"]'))
)
reset_filter.click()
self.logger.debug("Max Price Filter Selection Made: RESET.")

search_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="search-button"]'))
)
search_button.click()
self.logger.debug("Clicked Search Button.")

time.sleep(1)
```

Step 9: Select a Flight and Book Ticket:

This step selects the first item that appears in the search results, and proceeds to click the "Book My Ticket" that appears upon the selection.

```
#=== SELECT FLIGHT ===

flight_selection = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="search-result-0"]'))
)
flight_selection.click()
self.logger.debug("Clicked First Flight Result.")

#=== BOOK FLIGHT ====

buttons = self.driver.find_elements(By.CSS_SELECTOR, '[data-testid="search-button"]')
if len(buttons) > 1:
    book_ticket_button = buttons[1]
    book_ticket_button.click()
    self.logger.debug("Clicked Book My Ticket Now.")
else:
    print("Less than two buttons found with the specified data-testid.")
```

Seat Booking End-To-End Tests

Step 1: Open and navigate to the seat booking page

This involves logging in and running some of the previous tests so that the application state contains a selected flight that can be booked.

```
self.search_page_navigation()

# Assert we are on the seat booking page
WebDriverWait(self.driver, 10).until(
    lambda driver: '/seat-booking' in driver.current_url,
    message="We arent on the '/seat-booking' page"
)

# Ensure the seat booking page loaded
WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.CSS_SELECTOR, '[data-testid="seat-booking"]'))
)
self.logger.debug("Seat booking page loaded successfully")
```

Step 2: Select a seat for the first passenger

Click an available seat, fill in the passenger form, and verify the footer details are updated.

```
# Act: Select the first available seat
first_seat = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="seat-40-available"]'))
)
first_seat.click()
self.logger.debug("Selected the first available seat")

# Assert: Ensure the booking form is displayed
booking_form = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.CSS_SELECTOR, '[data-testid="booking-form-column"]'))
)
assert booking_form.is_displayed(), "Booking form was not displayed after selecting a seat"
self.logger.debug("Booking form displayed after selecting a seat")

# Fill out passenger details
self.fill_passenger_details(
    first_name="John",
    last_name="Doe",
    email="john.doe@email.com",
    phone="1234567890"
)
self.logger.debug("Filled out passenger details for the first passenger")

# Assert passenger details in the footer
self.verify_footer_details(seat_number="40", passenger_name="John Doe")
```

Step 3: Select the next passenger button

Click the next passenger button to close the form and select a seat for the second passenger.

```
# Act: Click the Next Passenger button
next_passenger_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="next-passenger-button"]'))
)
next_passenger_button.click()
self.logger.debug("Clicked the 'Next Passenger' button")
```

Step 4: Select a seat for the second passenger

Click an available seat, fill in the passenger form, and verify the footer details are updated.

```
# Act: Select the second available seat
second_seat = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="seat-41-available"]'))
)
second_seat.click()
self.logger.debug("Selected the second available seat")

# Fill out passenger details for the second passenger
self.fill_passenger_details(
    first_name="Jane",
    last_name="Smith",
    email="jane.smith@email.com",
    phone="0987654321"
)
self.logger.debug("Filled out passenger details for the second passenger")

# Assert passenger details in the footer
self.verify_footer_details(seat_number="41", passenger_name="Jane Smith")
```

Step 5: Select the book return flight button

Click the book return flight button to close the form and select a return flight seats for both passengers.

```
# Click Book Return Flight button
book_return_flight_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="book-return-flight-button"]'))
)
book_return_flight_button.click()
self.logger.debug("Clicked the 'Book Return Flight' button")
```

Step 6: Select a seat for the first passenger

Click an available seat and verify the form and footer details match the existing form entry.

```
# Act: Select return flight seats and verify passenger details
self.select_return_seat_and_verify(seat_number="54", passenger_name="John Doe")
self.verify_passenger_details(
    first_name="John",
    last_name="Doe",
    email="john.doe@email.com",
    phone="1234567890"
)
```

Step 7: Select the next passenger button

Click the next passenger button to close the form and select a return seat for the second passenger.

```
# Act: Click the Next Passenger button
next_passenger_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="next-passenger-button"]'))
)
next_passenger_button.click()
self.logger.debug("Clicked the 'Next Passenger' button")
```

Step 8: Select a seat for the second passenger

Click an available seat, fill in the passenger form, and verify the form and footer details match the existing form entries.

```
self.select_return_seat_and_verify(seat_number="55", passenger_name="Jane Smith")
self.verify_passenger_details(
    first_name="Jane",
    last_name="Smith",
    email="jane.smith@email.com",
    phone="0987654321"
)
```

Step 9: Select the checkout button to proceed to the payment page.

Click the checkout button and check that we are redirected.

```
# Act: Submit the seat booking form
checkout_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="checkout-button"]'))
)
checkout_button.click()
self.logger.debug("Clicked the 'Checkout' button")

# Assert: Ensure redirection to the checkout page
WebDriverWait(self.driver, 10).until(
    lambda driver: '/checkout' in driver.current_url,
    message="Redirection to '/checkout' did not occur"
)
self.logger.debug("Successfully redirected to the checkout page")
```

Login End-To-End Tests

Step 1: Open the home page

This ensures that the application home page is loaded.

```
self.driver.get("http://localhost:3000/")
self.logger.debug("Opened the application homepage.")

self.driver.set_window_size(968, 533)
self.logger.debug("Set the browser window size.")
```

Step 2: Click the login button

Click the login button in the navigation header.

```
login_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="login-button"]'))
)
login_button.click()
self.logger.debug("Clicked the login button.")
```

Step 3: Ensure the form is shown

Check that the email and password form fields are shown in the slideout.

```
email_input = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.ID, "email"))
)
email_input.send_keys("user@email.com")
self.logger.debug("Entered the email address.")

password_input = WebDriverWait(self.driver, 10).until(
    EC.presence_of_element_located((By.ID, "password"))
)
password_input.send_keys("password")
self.logger.debug("Entered the password.")
```

Step 4: Submit the form

Click the login button to submit the form with the default login values.

```
submit_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="submit-button"]'))
)
submit_button.click()
self.logger.debug("Submitted the login form.")
```

Step 5: Go to the Dashboard

Check that the user avatar is now shown, click it, and select the dashboard dropdown.

```
user_avatar = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="user-avatar"]'))
)
user_avatar.click()

dashboard_button = WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable((By.CSS_SELECTOR, '[data-testid="dashboard-button"]'))
)
dashboard_button.click()
```

Step 6: Confirm Dashboard redirect

Check that we have been redirected to the dashboard page.

```
WebDriverWait(self.driver, 10).until(
    lambda driver: '/dashboard' in driver.current_url,
    message="Redirection to '/dashboard' did not occur within the timeout period."
)

self.logger.debug("Successfully logged in and reached the dashboard.")
```