

Projekt zaliczeniowy, pn.

„Stworzyć zautomatyzowane testy, aby przyspieszyć przeprowadzanie testów aplikacji i zredukować testy manualne”

Uniwersytet WSB Merito Chorzów

Kierunek: Tester Oprogramowania

Piotr Staszak

Nr albumu 94191

Spis treści

1.	Wstęp	3
2.	Struktura projektu	3
3.	Opis testów	4
3.1.	Test 1: Przełączanie widoków mapy.....	5
3.2.	Test 2: Wyszukiwanie lokalizacji	7
3.3.	Test 3: Wyznaczanie trasy.....	9
3.4.	Test 4: Pomiar odległości.....	11
3.5.	Test 5: Walidacja autouzupełniania	13
3.6.	Test 6: Przełączanie trybów trasy	16
4.	Wdrożenie i integracja	19
5.	Podsumowanie.....	22

1. Wstęp

Celem projektu było stworzenie zestawu zautomatyzowanych testów funkcjonalnych dla aplikacji internetowej — interaktywnej aplikacji mapowej. W projekcie posłużyono się stroną internetową <https://mapy.com/pl/>. Automatyzacja testów ma na celu znaczące przyspieszenie procesu testowania oraz ograniczenie nakładu pracy wykonywanej ręcznie (testów manualnych).

Dzięki wykorzystaniu biblioteki Playwright oraz języka Python, testy symulują realne działania użytkownika w przeglądarce, co pozwala wykrywać błędy i regresje w aplikacji szybciej i z większą powtarzalnością niż w przypadku testów manualnych.

Projekt zawiera również integrację z systemem Continuous Integration GitHub Actions, co umożliwia regularne i automatyczne uruchamianie testów.

2. Struktura projektu

Projekt składa się z następujących elementów:

- modul_strona.py – moduł startujący stronę i konfigurujący sesję testową,
- test_1_przelaczanie_widokow.py – test sprawdzający przełączanie widoków mapy,
- test_2_sprawdzenie_lokalizacji.py – test wyszukiwania lokalizacji i geolokalizacji,
- test_3_sprawdzenie_trasy.py – test wyznaczania i wyświetlania trasy między punktami,
- test_4_sprawdzenie_pomiaru.py – test pomiaru odległości między punktami na mapie,
- test_5_walidacja_uzupełniania.py – test działania autouzupełniania w polu wyszukiwania,
- test_6_przelaczanie_trybow_trasy.py – test zmiany trybów wyznaczania trasy (samochód, pieszo, rower, transport publiczny),
- .github/workflows/testy.yml – konfiguracja automatycznego uruchamiania testów w GitHub Actions,
- projekt_zaliczeniowy_Piotr_Staszak.pdf – dokumentacja projektu.

3. Opis testów

Moduł modul_strona.py pełni funkcję inicjalizatora środowiska testowego dla wszystkich testów automatycznych. Jego głównym zadaniem jest uruchomienie przeglądarki w trybie headless, otwarcie nowego kontekstu i załadowanie testowanej strony internetowej. Dzięki temu każdy test zaczyna od gotowej, świeżej instancji strony, co zapewnia niezależność testów i powtarzalność wyników.

Funkcja odpalanie_strony zwraca dwie wartości: context (kontekst przeglądarki) oraz page (kartę/zakładkę), na której wykonywane są dalsze działania testowe.

Implementacja funkcji w modul_strona.py:

```
3  async def odpalanie_strony(p):
4      context = await p.chromium.launch_persistent_context(
5          user_data_dir="/tmp/playwright",
6          headless=True,
7          locale="pl-PL",
8          permissions=["geolocation"],
9          geolocation={"latitude": 51.107883, "longitude": 17.038538},
10         )
11
12     page = context.pages[0] if context.pages else await context.new_page()
13     await page.goto("https://mapy.com/pl/")
```

W kodzie testów importujemy ten moduł i wywołujemy funkcję odpalanie_strony, co pozwala skupić się w testach na logice sprawdzania funkcjonalności, a nie na konfiguracji środowiska.

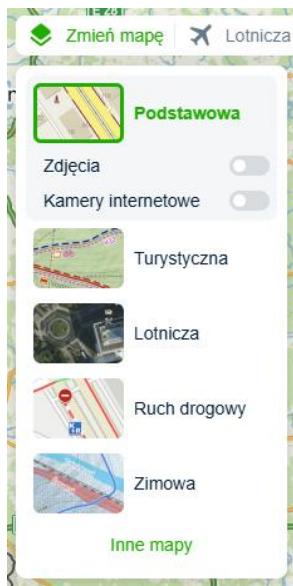
Zaletą takiego rozwiązania jest łatwość zarządzania konfiguracją uruchomienia przeglądarki (np. zmiana adresu URL testowanej aplikacji, czy ustawień przeglądarki) w jednym miejscu, bez konieczności edycji każdego testu osobno.

Po poprawnym wykonaniu testu powinniśmy otrzymać komunikat:

„Strona jest widoczna”.

3.1. Test 1: Przelaczanie widoków mapy

Test ten ma na celu weryfikację poprawnego działania przycisków zmiany widoku mapy. Sprawdza, czy użytkownik może przełączać się między różnymi warstwami mapy (podstawowa, turystyczna, lotnicza, drogowa, zimowa) oraz czy przełączniki są aktywne i reagują na kliknięcia.



Test potwierdza, że po każdej zmianie widok jest poprawnie renderowany i nie występują błędy ładowania.

Kod testu:

```
13  async def test_przelaczanie_widokow(page):
14      await page.wait_for_selector('mapy-map-toggle.map-controls__mapset')
15      await page.click('mapy-map-toggle.map-controls__mapset')
16
17      wszystkie_dzialaja = True
18
19      for url_fragment, polska_nazwa in WIDOKI.items():
20          try:
21              await page.wait_for_selector(f'img[src*="{url_fragment}-small.png"]', timeout=5000)
22              await page.click(f'img[src*="{url_fragment}-small.png"]')
23              await page.wait_for_timeout(1000)
24
25              current_url = page.url
26              if url_fragment in current_url:
27                  print(f"Mapa {polaska_nazwa} włącza się prawidłowo")
28              else:
29                  print(f"Mapa {polaska_nazwa} nie włączyła się poprawnie (URL: {current_url})")
30                  wszystkie_dzialaja = False
31          except Exception as e:
32              print(f"Błąd przy włączaniu widoku '{polaska_nazwa}': {e}")
33              wszystkie_dzialaja = False
34
35      if wszystkie_dzialaja:
36          print("Przełączanie wszystkich mapy działa prawidłowo")
37      else:
38          print("Nie wszystkie testy przeszły poprawnie")
```

Po poprawnym wykonaniu testu powinniśmy otrzymać komunikaty:

„Test bazowy przeszedł, uruchamiam test przełączania widoków”,

„Mapa turystyczna włącza się prawidłowo”,

„Mapa lotnicza włącza się prawidłowo”,

„Mapa drogowa włącza się prawidłowo”,

„Mapa zimowa włącza się prawidłowo”,

„Mapa podstawowa włącza się prawidłowo”,

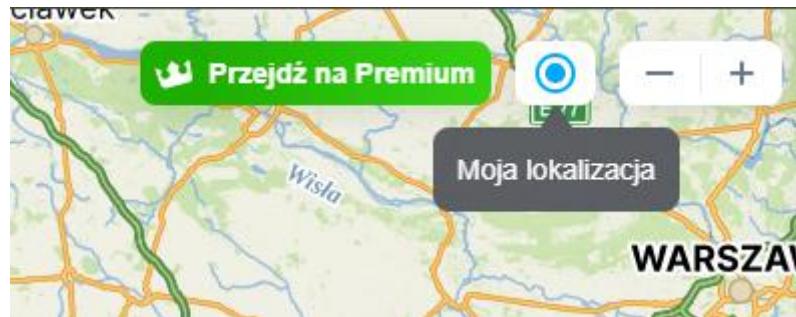
„Przełączanie wszystkich widoków mapy działa prawidłowo”.

3.2. Test 2: Wyszukiwanie lokalizacji

W tym teście automatycznym sprawdzamy, czy aplikacja poprawnie wyświetla zadaną lokalizację na mapie oraz czy adres URL zawiera prawidłowe współrzędne geograficzne odpowiadające tej lokalizacji.

Procedura testowa:

- Po załadowaniu strony czekamy na pojawienie się przycisku lokalizacji (np. ikony geolokalizacji na mapie),



- Klikamy ten przycisk, co powinno spowodować zlokalizowanie użytkownika lub ustawienie mapy na konkretną lokalizację,
- Następnie test czeka na pojawienie się na mapie wizualnego znacznika lokalizacji (np. punktu lub pinezki),
- Na koniec weryfikuje, czy aktualny URL strony zawiera oczekiwane współrzędne geograficzne (np. "x=17.0385380" i "y=51.1078830" dla Wrocławia).

Kod testu:

```
5  async def test_moja_lokalizacja(page):
6      selector = 'mapy-map-button.map-controls__geolocation'
7      await page.wait_for_selector(selector, timeout=5000)
8
9      lokalizacja_button = await page.query_selector(selector)
10     if lokalizacja_button and await lokalizacja_button.is_enabled():
11         await lokalizacja_button.click()
12         print("Kliknięto przycisk lokalizacji")
13     else:
14         print("Nie można kliknąć przycisku lokalizacji")
15         return
16
17     await page.wait_for_timeout(3000)
18
19     try:
20         await page.wait_for_selector('.geolocation-mark', timeout=5000)
21         print("Znacznik lokalizacji pojawił się na mapie")
22     except:
23         print("Znacznik lokalizacji NIE pojawił się na mapie")
24
25     url = page.url
26     print(f"Obecny URL: {url}")
27     if "x=17.0385380" in url and "y=51.1078830" in url:
28         print("URL zawiera współrzędne lokalizacji (Wrocław) i cały test przebiegł prawidłowo")
29     else:
30         print("URL nie zawiera spodziewanych współrzędnych")
31
32     print("Test lokalizacji zakończony")
```

Dzięki temu testowi upewniamy się, że aplikacja nie tylko reaguje na żądanu lokalizacji, ale także że pozycja jest prawidłowo ustawiana i wyświetlana, a link do mapy zawiera dokładne dane GPS.

Po poprawnym wykonaniu testu powinniśmy otrzymać komunikaty:

„Kliknięto przycisk lokalizacji”,

„Znacznik lokalizacji pojawił się na mapie”,

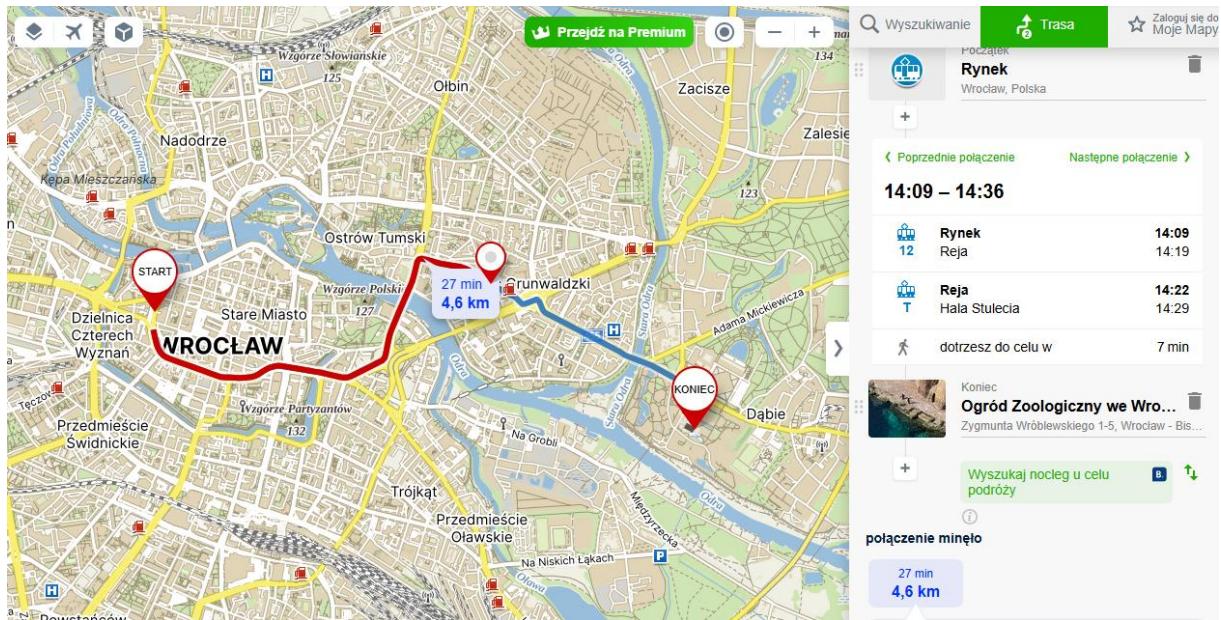
„Obecny URL: <https://mapy.com/pl/zakladni?x=17.0385380&y=51.1078830&z=18>”,

„URL zawiera współrzędne lokalizacji (Wrocław) i cały test przebiegł prawidłowo”,

„Test lokalizacji zakończony”.

3.3. Test 3: Wyznaczanie trasy

Test ten wprowadza dwa punkty na mapie („Zoo Wrocław” jako cel i „Rynek Wrocław” jako punkt startowy) i sprawdza, czy aplikacja poprawnie wyznacza trasę między nimi. Weryfikuje, czy na stronie pojawia się informacja o długości trasy oraz czy użytkownik może zobaczyć szczegóły dotyczące przebiegu trasy. Test ten jest kluczowy dla funkcjonalności planowania podróży i musi potwierdzić, że trasy generowane są bez błędów.



Kod testu:

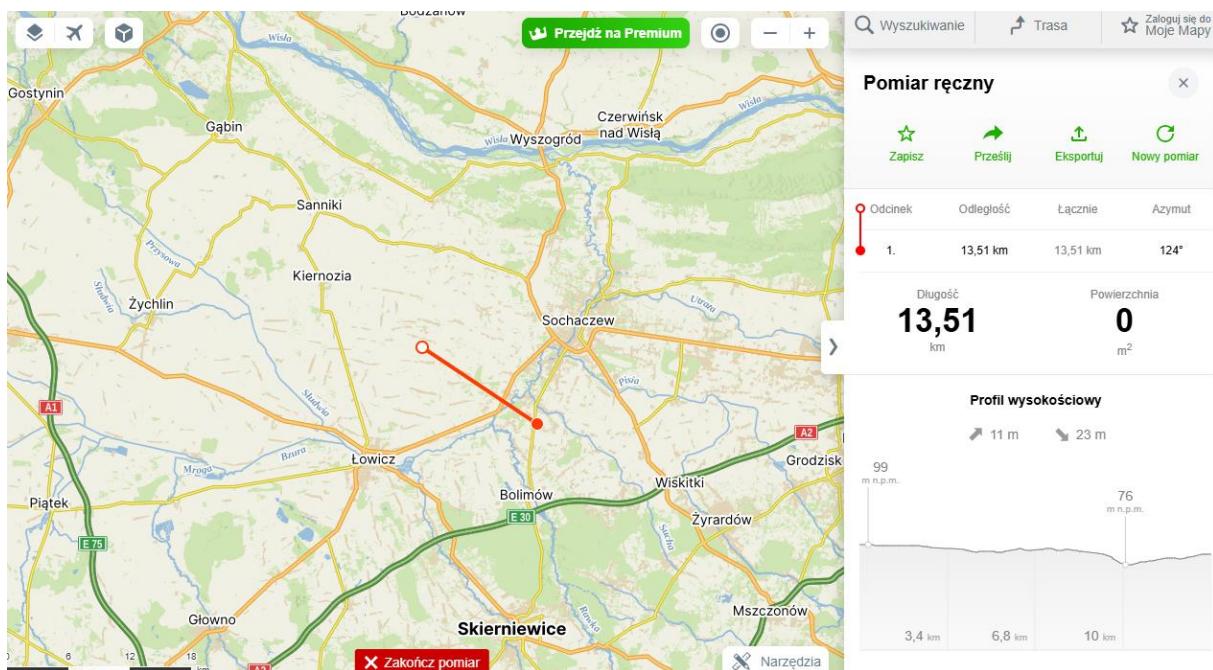
```
5  async def test_wyszukiwanie_trasy(page):
6      await page.wait_for_selector('#search > form > input[type="text"]', timeout=7000)
7      await page.fill('#search > form > input[type="text"]', 'Zoo Wrocław')
8      print("Wpisano 'Zoo Wrocław''")
9
10     await page.click('#search > form > button.submit.notranslate')
11     print("Kliknięto ikonę lupy")
12
13     await page.wait_for_selector('a:has-text("Trasa")', timeout=10000)
14     await page.click('a:has-text("Trasa")')
15     print("Kliknięto przycisk 'Trasa''")
16
17     await page.wait_for_selector('input[placeholder="Wskaż początek"]', timeout=7000)
18     await page.fill('input[placeholder="Wskaż początek"]', 'Rynek Wrocław')
19     print("Wpisano 'Rynek Wrocław' jako start")
20
21     await page.wait_for_selector('div.route-items-list.unrouted ul li:nth-child(1)', timeout=7000)
22     await page.click('div.route-items-list.unrouted ul li:nth-child(1)')
23     print("Kliknięto pierwszy element na liście tras")
24
25     await page.wait_for_selector('span.distance', timeout=10000)
26     distance_text = await page.inner_text('span.distance')
27     print("Wyznaczona trasa ma długość:", distance_text)
```

Po poprawnym wykonaniu testu powinniśmy otrzymać komunikaty:

- „Wpisano 'Zoo Wrocław'”,
- „Kliknięto ikonę lupy”,
- „Kliknięto przycisk 'Trasa'”,
- „Wpisano 'Rynek Wrocław' jako start”,
- „Kliknięto pierwszy element na liście tras”,
- „Wyznaczona trasa ma długość: 4,6 km”.

3.4. Test 4: Pomiar odległości

Test mierzy odległość między dwoma punktami wybranymi na mapie przy użyciu wbudowanego narzędzia pomiaru. Sprawdza, czy narzędzie jest dostępne po wybraniu odpowiedniej funkcji w menu i czy po kliknięciu dwóch punktów na mapie wyświetla się poprawny dystans wraz z jednostką miary. Test symuluje też interakcję użytkownika (powiększanie mapy) i potwierdza, że narzędzie działa niezależnie od zoomu.



Kod testu:

```
29 |     await page.wait_for_selector('#map > div:nth-child(1) > svg', timeout=7000)
30 |     svg = await page.query_selector('#map > div:nth-child(1) > svg')
31 |     box = await svg.bounding_box()
32 |     if box:
33 |         mid_x = box["width"] / 2
34 |         mid_y = box["height"] / 2
35 |         await svg.click(position={"x": mid_x, "y": mid_y})
36 |         await pause()
37 |         await svg.click(position={"x": mid_x + 120, "y": mid_y + 80})
38 |         print("Kliknięto dwa punkty na mapie")
39 |         await pause()
40 |
41 |     await page.wait_for_selector('#distance-meter > div > div.scroll-section > ul > li.distance > strong', timeout=7000)
42 |     dystans = await page.inner_text('#distance-meter > div > div.scroll-section > ul > li.distance > strong')
43 |     jednostka = await page.inner_text('#distance-meter > div > div.scroll-section > ul > li.distance > small.unit')
44 |     print(f"Zmierzona odległość to: {dystans} {jednostka}")
45 |     await pause(2000)
```

```
8  async def test_mierzenie_odleglosci(page):
9      await page.wait_for_selector('#map-controls > div.map-controls__bottomToolbar > mapy-map-toggle', timeout=7000)
10     await page.click('#map-controls > div.map-controls__bottomToolbar > mapy-map-toggle')
11     print("Kliknięto ikonę narzędzi")
12     await pause()
13
14     await page.wait_for_selector(
15         'body > div.ui-popover.toolsMenu__popOver > mapy-mapmenu > mapy-mapmenu-item:nth-child(7)',
16         timeout=7000
17     )
18     await page.click(
19         'body > div.ui-popover.toolsMenu__popOver > mapy-mapmenu > mapy-mapmenu-item:nth-child(7)'
20     )
21     print("Wybrano 'Mierzenie odległości'")
22     await pause()
23
24     for _ in range(2):
25         await page.mouse.wheel(delta_x=0, delta_y=-300)
26         await pause(500)
27     print("Powiększono mapę")
```

Po poprawnym wykonaniu testu powinniśmy otrzymać komunikaty:

„Kliknięto ikonę narzędzi”,

„Wybrano 'Mierzenie odległości'”,

„Powiększono mapę”,

„Kliknięto dwa punkty na mapie”,

„Zmierzona odległość to: 13,51 km”.

3.5. Test 5: Walidacja autouzupełniania

Test sprawdza działanie mechanizmu autouzupełniania w polu tekstowym wyszukiwarki. Automatycznie wpisuje frazy (np. „Zoo”, „Zoo Wrocław”) i weryfikuje, czy lista wyników dynamicznie się aktualizuje oraz czy różne zapytania skutkują różnymi wynikami. Test porównuje listy wyników dla różnych fraz, co pozwala wykryć potencjalne problemy z filtrowaniem i poprawnością podpowiedzi. W celu uproszczenia logiki, test przyjmuje za pozytywny przypadek sytuację, gdy występują różnice pomiędzy listami wyników dla dwóch fraz. W rzeczywistości nie każda zmiana zapytania musi skutkować innym wynikiem (np. synonimy, różne formy gramatyczne). Test nie sprawdza trafności wyników (np. odległości geograficznej), jedynie zmienność listy podpowiedzi.

The image displays two side-by-side screenshots of a search interface, likely from a mobile application or website. Both screens feature a green header bar with a magnifying glass icon labeled "Wyszukiwanie", a "Trasa" button with a map pin icon, and a "Zaloguj się do Moje Mapy" button with a star icon.

Left Screenshot (Search for "Zoo"): The search bar contains the text "Zoo". Below it, a list of suggestions is shown:

- Zoo** Kategoria
- Zoovu** Usługi dla firm, Rynek 39/40, Wrocław - Stare Miasto, ...
- Sklep zoologiczny - sklep dla psa, kota i - H...** Artykuły dla zwierząt, Piaskowa 17, Wrocław - Stare Mi...
- Ogród Zoologiczny we Wrocławiu** Zoo, Zygmunta Wróblewskiego 1-5, Wrocław - Biskupi...
- Muzeum Przyrodnicze we Wrocławiu** Muzeum, Henryka Sienkiewicza 21, Wrocław - Stare M...
- Zoo Team** Mini zoo, Na Szańcach 7a, Wrocław - Ołbin, Polska
- Hotel Zoo by Afrykarium Wrocław** Wróblewskiego 7, Wrocław - Biskupin-Sępolno-Dąbie-...
- Maxi Zoo** Plac Grunwaldzki 22, Wrocław - Plac Grunwaldzki, Pol...

Right Screenshot (Search for "Zoo Wrocław"): The search bar now contains the text "Zoo Wrocław". The list of suggestions has changed:

- Ogród Zoologiczny we Wrocławiu** Zoo, Zygmunta Wróblewskiego 1-5, Wrocław - Biskupi...
- Wróblewskiego (ZOO)** Wrocław - Biskupin-Sępolno-Dąbie-Bartoszowice, Polska

Kod testu:

```
5  async def wpisz_i_pobierz_wyniki(page, zapytanie: str):
6      await page.fill('#search > form > input[type="text"]', '')
7
8      poprzedni_tekst = ''
9      pierwszy_wynik = await page.query_selector('li.item span.text > strong')
10     if pierwszy_wynik:
11         poprzedni_tekst = (await pierwszy_wynik.inner_text()).strip()
12
13     await page.type('#search > form > input[type="text"]', zapytanie, delay=100)
14
15     try:
16         await page.wait_for_function(
17             f'document.querySelector("li.item span.text > strong") && ' +
18             f'document.querySelector("li.item span.text > strong").innerText.trim() != "{poprzedni_tekst}"',
19             timeout=10000
20         )
21     except TimeoutError:
22         print(f"Lista wyników dla '{zapytanie}' nie zmieniła się w czasie 10 sekund.")
23         return []
24
25     elementy = await page.query_selector_all('li.item')
26     wyniki = []
27     for el in elementy:
28         strong = await el.query_selector('span.text > strong')
29         nazwa = (await strong.inner_text()).strip() if strong else ''
30         em = await el.query_selector('span.text > em')
31         adres = (await em.inner_text()).strip() if em else ''
32         wyniki.append((nazwa, adres))
33
34     return wyniki
```

```
36  async def main():
37      async with async_playwright() as p:
38          context, page = await odpalanie_strony(p)
39
40          try:
41              wyniki1 = await wpisz_i_pobierz_wyniki(page, "Zoo")
42              print(f"Znaleziono {len(wyniki1)} wyników dla 'Zoo':")
43              for i, (nazwa, adres) in enumerate(wyniki1, 1):
44                  print(f"{i}. {nazwa} - {adres}")
45
46              wyniki2 = await wpisz_i_pobierz_wyniki(page, "Zoo Wrocław")
47              print(f"\nZnaleziono {len(wyniki2)} wyników dla 'Zoo Wrocław':")
48              for i, (nazwa, adres) in enumerate(wyniki2, 1):
49                  print(f"{i}. {nazwa} - {adres}")
50
51              set1 = set(wyniki1)
52              set2 = set(wyniki2)
53              wspolne = set1.intersection(set2)
54              tylko_wyniki1 = set1 - set2
55              tylko_wyniki2 = set2 - set1
56
57              print(f"\nLiczba wyników wspólnych: {len(wspolne)}")
58              print(f'Liczba wyników tylko dla "Zoo": {len(tylko_wyniki1)}')
59              print(f'Liczba wyników tylko dla "Zoo Wrocław": {len(tylko_wyniki2)}')
60
61              if len(tylko_wyniki1) > 0 or len(tylko_wyniki2) > 0:
62                  print("\n⚠ Wyniki wyszukiwania różnią się między frazami, co oznacza, że strona poprawnie wyszukuje różne miejsca.")
63              else:
64                  print("\n⚠ Wyniki dla obu fraz są identyczne. Może to oznaczać, że wyszukiwarka na stronie nie działa prawidłowo lub frazy nie są rozróżniane.")
65
66          finally:
67              await context.close()
```

Po poprawnym wykonaniu testu powinniśmy otrzymać komunikaty:

„Znaleziono $\{len(wyniki1)\}$ wyników dla 'Zoo'" po którym zostaną wyszczególnione wszystkie wyniki;

„Znaleziono $\{len(wyniki2)\}$ wyników dla 'Zoo Wrocław':" po którym zostaną wyszczególnione wszystkie wyniki;

„Liczba wyników wspólnych: $\{len(wspolne)\}\",$

„Liczba wyników tylko dla 'Zoo': $\{len(tylko_wyniki1)\}\",$

„Liczba wyników tylko dla 'Zoo Wrocław': $\{len(tylko_wyniki2)\}\",$

„ Wyniki wyszukiwania różnią się między frazami, co oznacza, że strona poprawnie wyszukuje różne miejsca.”.

Przykładowy poprawny wynik testu:

```
Znaleziono 8 wyników dla 'Zoo':
1. zoo - Kategoria
2. Zoovu - Usługi dla firm, Rynek 39/40, Wrocław - Stare Miasto, Polska
3. Sklep zoologiczny - sklep dla psa, kota i - Hala Targowa Wrocław - Artykuły dla zwierząt, Piaskowa 17, Wrocław - Stare Miasto, Polska
4. Ogród Zoologiczny we Wrocławiu - Zoo, Zygmunta Wróblewskiego 1-5, Wrocław - Biskupin-Sępolno-Dąbie-Bartoszowice, Polska
5. Muzeum Przyrodnicze we Wrocławiu - Muzeum, Henryka Sienkiewicza 21, Wrocław - Stare Miasto, Polska
6. Zoo Team - Mini zoo, Na Szaficach 7a, Wrocław - Ołbin, Polska
7. Hotel Zoo by Afrykanium Wrocław - Wróblewskiego 7, Wrocław - Biskupin-Sępolno-Dąbie-Bartoszowice, Polska
8. Maxi Zoo - Plac Grunwaldzki 22, Wrocław - Plac Grunwaldzki, Polska

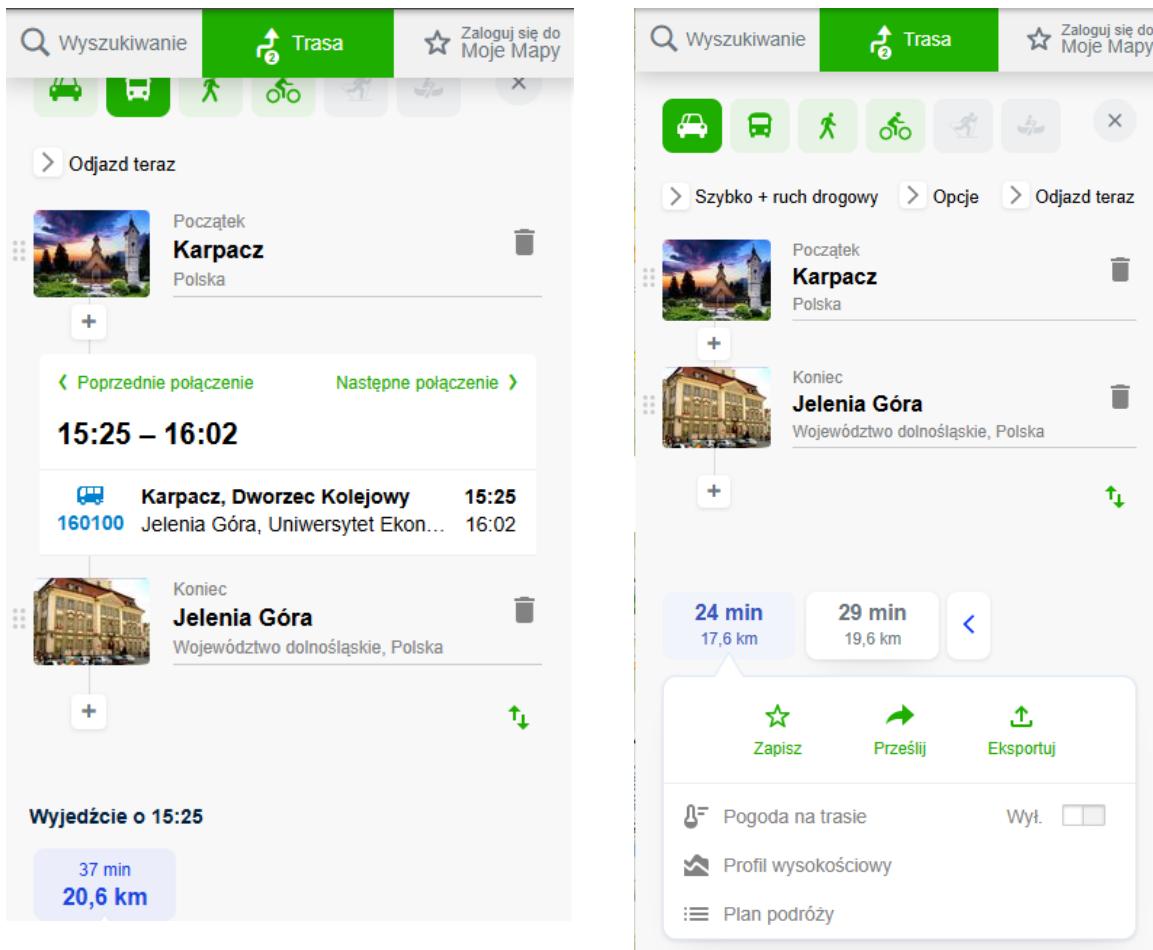
Znaleziono 2 wyników dla 'Zoo Wrocław':
1. Ogród Zoologiczny we Wrocławiu - Zoo, Zygmunta Wróblewskiego 1-5, Wrocław - Biskupin-Sępolno-Dąbie-Bartoszowice, Polska
2. Wróblewskiego (ZOO) - Wrocław - Biskupin-Sępolno-Dąbie-Bartoszowice, Polska

Liczba wyników wspólnych: 1
Liczba wyników tylko dla 'Zoo': 7
Liczba wyników tylko dla 'Zoo Wrocław': 1

 Wyniki wyszukiwania różnią się między frazami, co oznacza, że strona poprawnie wyszukuje różne miejsca.
```

3.6. Test 6: Przelaczanie trybów trasy

Ten test weryfikuje poprawność działania przycisków przełączających tryb wyznaczania trasy (np. samochodem, pieszo, rowerem, transportem publicznym). Dla każdego trybu test odczytuje wyświetląną długość trasy i porównuje ją z poprzednią, aby upewnić się, że trasy faktycznie się różnią. Jest to ważne, ponieważ różne tryby mogą mieć różne ograniczenia i przebieg tras (np. pieszo może być krótsza ścieżka, niedostępna dla samochodów).



Kod testu:

```
5  async def test_porownanie_trybow_po_wyszukaniu(page):
6      await page.wait_for_selector('#search > form > input[type="text"]', timeout=7000)
7      await page.fill('#search > form > input[type="text"]', 'Jelenia Góra')
8      print("Wpisano 'Jelenia Góra'")
9
10     await page.click('#search > form > button.submit.notranslate')
11     print("Kliknięto ikonę lupy")
12
13     await page.wait_for_selector('a:has-text("Trasa")', timeout=10000)
14     await page.click('a:has-text("Trasa")')
15     print("Kliknięto przycisk 'Trasa'")
16
17     await page.wait_for_selector('input[placeholder="Wskaż początek"]', timeout=7000)
18     await page.fill('input[placeholder="Wskaż początek"]', 'Karpacz')
19     print("Wpisano 'Karpacz' jako start")
20
21     await page.wait_for_selector('div.route-items-list.unrouted ul li:nth-child(1)', timeout=7000)
22     await page.click('div.route-items-list.unrouted ul li:nth-child(1)')
23     print("Kliknięto pierwszy element na liście tras")
24
25     await page.wait_for_selector('span.distance', timeout=10000)
26     dystans = await page.inner_text('span.distance')
27     print("Wyznaczona trasa ma długość:", dystans)
28
29     tryby = [
30         "Samochodem",
31         "Pieszo",
32         "Rowerem",
33         "Transportem publicznym"
34     ]
35
36     await page.wait_for_selector('button:has(svg > title)', timeout=10000)
37
38     poprzedni_dystans = None
39     test_udany = False
40
41     for tryb in tryby:
42         selector = f'button:has(svg > title:text("{tryb}"))'
43         print(f"Klikam tryb: {tryb}")
44         await page.click(selector)
45
46         await page.wait_for_selector('span.distance', timeout=10000)
47         dystans_trybu = await page.inner_text('span.distance')
48         print(f"Dystans dla trybu '{tryb}': {dystans_trybu}")
49
50         if poprzedni_dystans is not None:
51             if dystans_trybu == poprzedni_dystans:
52                 print(f"Uwaga! Dystans dla trybu '{tryb}' jest taki sam jak poprzedni: {dystans_trybu}")
53             else:
54                 print(f"Dystans dla trybu '{tryb}' różni się od poprzedniego.")
55             test_udany = True
56         poprzedni_dystans = dystans_trybu
57
58         await page.wait_for_timeout(1000)
59
60     if test_udany:
61         print("Test przeszedł pomyślnie – trasy dla różnych trybów się różnią.")
62     else:
63         print("Test NIE przeszedł – dystanse dla wszystkich trybów są takie same.")
64
65     async def main():
66         async with async_playwright() as p:
67             context, page = await odpalanie_strony(p)
68             try:
69                 await test_porownanie_trybow_po_wyszukaniu(page)
70             finally:
71                 await context.close()
```

Po poprawnym wykonaniu testu powinniśmy otrzymać komunikaty:

„Wpisano 'Jelenia Góra'”,

„Kliknięto ikonę lupy”,

„Kliknięto przycisk 'Trasa'”,

„Wpisano 'Karpacz' jako start”,

„Kliknięto pierwszy element na liście tras”,

„Wyznaczona trasa ma długość: 20,6 km”,

„Klikam tryb: Samochodem”,

„Dystans dla trybu 'Samochodem': 17,6 km”,

„Klikam tryb: Pieszo”,

„Dystans dla trybu 'Pieszo': 21 km”,

„Dystans dla trybu 'Pieszo' różni się od poprzedniego.”,

„Klikam tryb: Rowerem”,

„Dystans dla trybu 'Rowerem': 16,4 km”,

„Dystans dla trybu 'Rowerem' różni się od poprzedniego.”,

„Klikam tryb: Transportem publicznym”,

„Dystans dla trybu 'Transportem publicznym': 20,6 km”,

„Dystans dla trybu 'Transportem publicznym' różni się od poprzedniego.”,

„Test przeszedł pomyślnie — trasy dla różnych trybów się różnią.”.

4. Wdrożenie i integracja

Cały zestaw testów jest automatycznie uruchamiany na serwerach GitHub w ramach workflow skonfigurowanego w pliku YAML. Testy uruchamiane są codziennie o 7:00 czasu lokalnego oraz na żądanie (manualne wywołanie). W przypadku niepowodzenia któregokolwiek testu, system automatycznie wysyła powiadomienie e-mail na wskazany adres. Dzięki temu możliwe jest szybkie reagowanie na regresje i problemy z aplikacją. Dzięki temu proces testowy odbywa się regularnie bez potrzeby ręcznego uruchamiania. W razie wykrycia błędów system wysyła powiadomienia mailowe do zespołu.

Kod workflow:

```
1   name: Testy automatyczne codziennie o 7 rano
2
3   on:
4     schedule:
5       - cron: '5 5 * * *' # codziennie o 5:00 UTC (7:00 CEST, czas letni)
6       workflow_dispatch: # umożliwia ręczne uruchomienie
7
8   jobs:
9     testy:
10       runs-on: ubuntu-latest
11
12     steps:
13       - uses: actions/checkout@v3
14
15       - name: Ustaw Python
16         uses: actions/setup-python@v4
17         with:
18           python-version: '3.13'
19
20       - name: Zainstaluj wymagane pakiety
21         run: |
22           python -m pip install --upgrade pip
23           pip install playwright
24           playwright install
25
26       - name: Uruchom testy
27         run: |
28           echo "::group::TEST 1: Przełączanie widoków"
29           python test_1_przelaczanie_widokow.py
30           echo "::endgroup::"
31
32           echo "::group::TEST 2: Sprawdzenie lokalizacji"
33           python test_2_sprawdzenie_lokalizacji.py
34           echo "::endgroup::"
35
36           echo "::group::TEST 3: Sprawdzenie trasy"
37           python test_3_sprawdzenie_trasy.py
38           echo "::endgroup::"
39
40           echo "::group::TEST 4: Sprawdzenie pomiaru"
41           python test_4_sprawdzenie_pomiaru.py
42           echo "::endgroup::"
```

```

44      echo "::group::TEST 5: Walidacja uzupełniania"
45      python test_5_walidacja_uzupełniania.py
46      echo "::endgroup::"
47
48      echo "::group::TEST 6: Przełączanie trybów trasy"
49      python test_6_przelaczanie_trybow_trasy.py
50      echo "::endgroup::"
51
52      - name: Wyślij mail w razie błędu
53        if: failure()
54        uses: dawidd6/action-send-mail@v3
55        with:
56          server_address: smtp.gmail.com
57          server_port: 587
58          username: ${{ secrets.SMTP_USERNAME }}
59          password: ${{ secrets.SMTP_PASSWORD }}
60          subject: "✖ Błąd testów automatycznych"
61          body: "Jeden lub więcej testów zakończył się błędem. Sprawdź wyniki w zakładce Actions na GitHubie."
62          to: piotrstaszak.jg@gmail.com
63          from: github-actions@example.com

```

Przykładowy wynik testu:

← Testy automatyczne codziennie o 7 rano

Testy automatyczne codziennie o 7 rano #28

Summary

Jobs

testy

Run details

Usage

Workflow file

testy

succeeded 7 hours ago in 2m 0s

- > Set up job
- > Run actions/checkout@v3
- > Ustaw Python
- > Zainstaluj wymagane pakiety
- > Uruchom testy

```

1 ► Run echo "::group::TEST 1: Przełączanie widoków"
33 ► TEST 1: Przełączanie widoków
43 ► TEST 2: Sprawdzenie lokalizacji
51 ► TEST 3: Sprawdzenie trasy
60 ► TEST 4: Sprawdzenie pomiaru
68 ► TEST 5: Walidacja uzupełniania
90 ► TEST 6: Przełączanie trybów trasy

```

- Wyślij mail w razie błędu
- > Post Ustaw Python
- > Post Run actions/checkout@v3
- > Complete job

Uruchom testy

```

1 ► Run echo "::group::TEST 1: Przełączanie widoków"
33 ▼ TEST 1: Przełączanie widoków
34 Przycisk cookies został kliknięty i zniknął
35 Strona jest widoczna
36 Test bazowy przeszedł, uruchamiam test przełączania widoków
37 Mapa turystyczna włącza się prawidłowo
38 Mapa lotnicza włącza się prawidłowo
39 Mapa drogowa włącza się prawidłowo
40 Mapa zimowa włącza się prawidłowo
41 Mapa podstawowa włącza się prawidłowo
42 Przełączanie wszystkich widoków mapy działa prawidłowo
43 ▼ TEST 2: Sprawdzenie lokalizacji
44 Przycisk cookies nie był widoczny, pomijam
45 Strona jest widoczna
46 Kliknięto przycisk lokalizacji
47 Znacznik lokalizacji pojawił się na mapie
48 Obecny URL: https://mapy.com/pl/zakladni?x=17.0385380&y=51.1078830&z=18
49 URL zawiera współrzędne lokalizacji (Wrocław) i cały test przebiegł prawidłowo
50 Test lokalizacji zakończony
51 ▼ TEST 3: Sprawdzenie trasy
52 Przycisk cookies nie był widoczny, pomijam
53 Strona jest widoczna
54 Wpisano 'Zoo Wrocław'
55 Kliknięto ikonę lupy
56 Kliknięto przycisk 'Trasa'
57 Wpisano 'Rynek Wrocław' jako start
58 Kliknięto pierwszy element na liście tras
59 Wyznaczona trasa ma długość: 4,9 km
60 ▼ TEST 4: Sprawdzenie pomiaru
61 Przycisk cookies nie był widoczny, pomijam
62 Strona jest widoczna
63 Kliknięto ikonę narzędzi
64 Wybrano 'Mierzenie odległości'
65 Powiększono mapę
66 Kliknięto dwa punkty na mapie
67 Zmierzona odległość to: 13,51 km
68 ► TEST 5: Walidacja uzupełniania
69 ► TEST 6: Przełączanie trybów trasy

```

Testy automatyczne codziennie o 7 rano			
		Event	Status
25 workflow runs		Event	Status
	This workflow has a <code>workflow_dispatch</code> event trigger.		
✓ Testy automatyczne codziennie o 7 rano	Testy automatyczne codziennie o 7 rano #28: Scheduled	main	2 hours ago ⌚ 2m 6s
✓ Testy automatyczne codziennie o 7 rano	Testy automatyczne codziennie o 7 rano #27: Scheduled	main	yesterday ⌚ 1m 52s
✓ Testy automatyczne codziennie o 7 rano	Testy automatyczne codziennie o 7 rano #26: Scheduled	main	2 days ago ⌚ 1m 43s
✓ Testy automatyczne codziennie o 7 rano	Testy automatyczne codziennie o 7 rano #25: Scheduled	main	3 days ago ⌚ 1m 41s
✓ Testy automatyczne codziennie o 7 rano	Testy automatyczne codziennie o 7 rano #24: Scheduled	main	4 days ago ⌚ 1m 52s
✓ Testy automatyczne codziennie o 7 rano	Testy automatyczne codziennie o 7 rano #23: Scheduled	main	5 days ago ⌚ 2m 5s
✓ Testy automatyczne codziennie o 7 rano	Testy automatyczne codziennie o 7 rano #22: Scheduled	main	last week ⌚ 1m 55s
✓ Testy automatyczne codziennie o 7 rano	Testy automatyczne codziennie o 7 rano #21: Scheduled	main	last week ⌚ 1m 52s
✓ Testy automatyczne codziennie o 7 rano	Testy automatyczne codziennie o 7 rano #20: Scheduled	main	last week ⌚ 1m 42s
✓ Testy automatyczne codziennie o 7 rano	Testy automatyczne codziennie o 7 rano #19: Scheduled	main	last week ⌚ 1m 56s

5. Podsumowanie

Stworzenie zestawu zautomatyzowanych testów znacząco usprawniło proces kontroli jakości aplikacji. Automatyzacja pozwoliła na:

- skrócenie czasu wykonywania testów,
- wyeliminowanie błędów ludzkich związanych z testami manualnymi,
- ciągłe monitorowanie stabilności aplikacji w czasie rozwoju,
- szybką detekcję regresji i błędów funkcjonalnych.

Projekt demonstruje praktyczne zastosowanie narzędzi do automatyzacji testów oraz ich integrację z systemami CI/CD, co stanowi cenną umiejętność w nowoczesnym procesie wytwarzania oprogramowania.