

# 일정 관리 시스템 UI 자동화



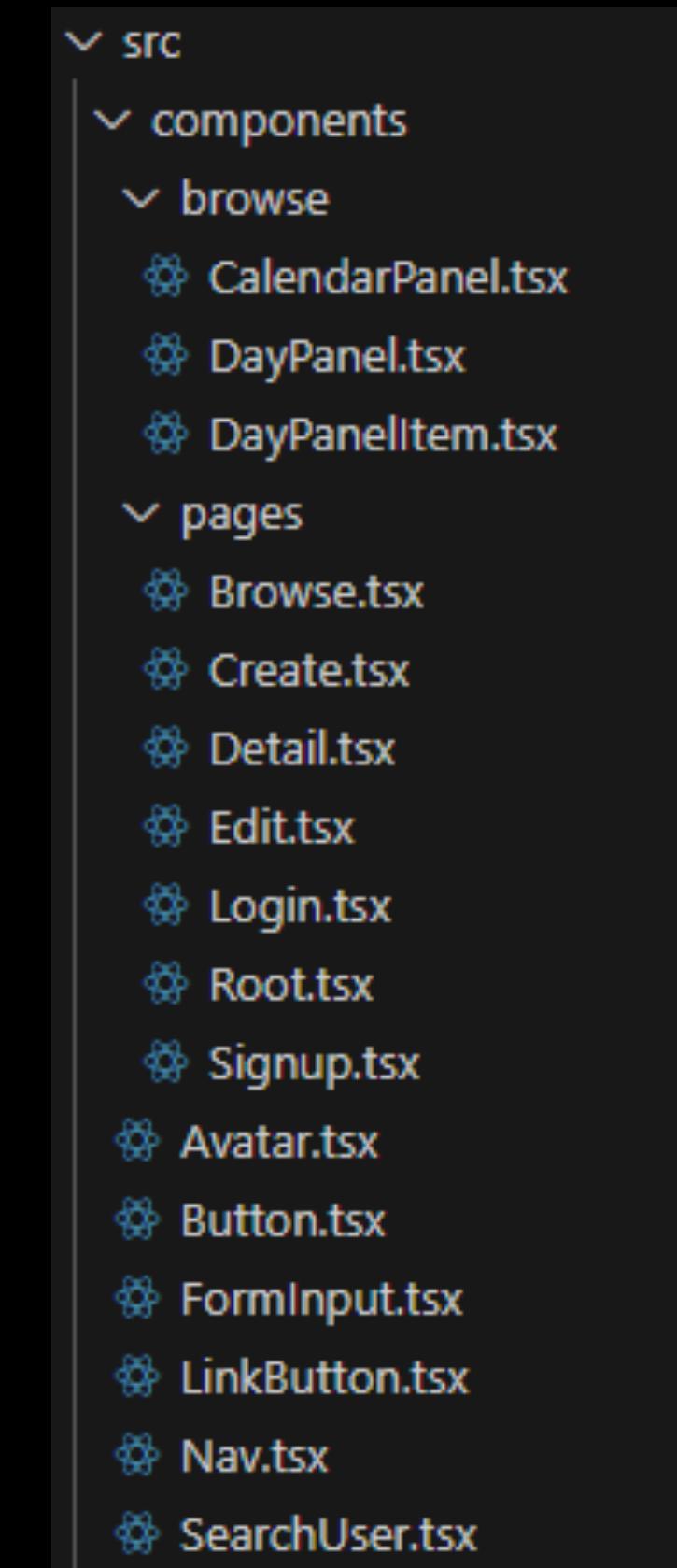
## 수업내용

- React 프로젝트의 구조를 이해하고, Selenium을 활용하여 주요 기능을 자동화 테스트합니다.
- 로그인, 회원가입, 검색, CRUD(생성, 조회, 수정, 삭제) 기능을 자동화하여 검증할 수 있습니다.
- 비동기 데이터 로딩과 UI 변경 감지를 처리하는 방법을 익힙니다.
- Selenium을 활용하여 동적 요소를 조작하고, 웹 애플리케이션의 반응성을 테스트할 수 있습니다.
- 테스트 실행 결과를 로그에 기록하고, 로그를 기반으로 문제를 분석합니다.

## Selenium 테스트 대상 폴더

- **pages/** → 사용자의 주요 기능을 담당하는 화면
- **components/** → 버튼, 입력 필드 등 재사용되는 UI 요소

- **Browse.tsx** → 일정 탐색 페이지 (캘린더 + 일정 목록 출력)
- **Create.tsx** → 새로운 일정을 추가하는 페이지
- **Detail.tsx** → 개별 일정 상세 페이지
- **Edit.tsx** → 일정 수정 페이지
- **Login.tsx** → 로그인 페이지
- **Signup.tsx** → 회원가입 페이지
- **Root.tsx** → 프로젝트의 최상위 페이지





# Selenium이 테스트할 주요 기능

일	월	화	수	목	금	토
						1
2	3	4 dsdd	5 afdf	6	7	8
9	10	11	12	13	14	15

## Selenium 자동화 테스트 적용 가능 기능

- 로그인 (Login.tsx) → 아이디/비밀번호 입력 후 버튼 클릭
- 회원가입 (Signup.tsx) → 사용자 정보 입력 후 가입 버튼 클릭
- 검색 (SearchUser.tsx) → 키워드 입력 후 검색 결과 확인
- CRU 기능
  - ✓ Create (Create.tsx) → 새 일정 입력 및 저장
  - ✓ Read (Detail.tsx) → 특정 일정 조회
  - ✓ Update (Edit.tsx) → 기존 일정 수정

## 테스트할 기능과 기존 학습 내용 연결



**send\_keys()** : 입력 필드 자동 입력



**click()** : 버튼 클릭 테스트



**find\_element()** : 특정 요소 찾기



**assert** : 결과 검증



**execute\_script()** : UI 변경 테스트

## Selenium 동작 방식

1. WebDriver → 브라우저를 제어하는 역할 (Chrome, Edge, Firefox 등)
2. find\_element() → 특정 HTML 요소를 선택하는 기능
3. send\_keys() → 입력 필드에 자동으로 값 입력
4. click() → 버튼을 클릭하는 동작 실행



```
from selenium import webdriver

# WebDriver 실행
driver = webdriver.Chrome()

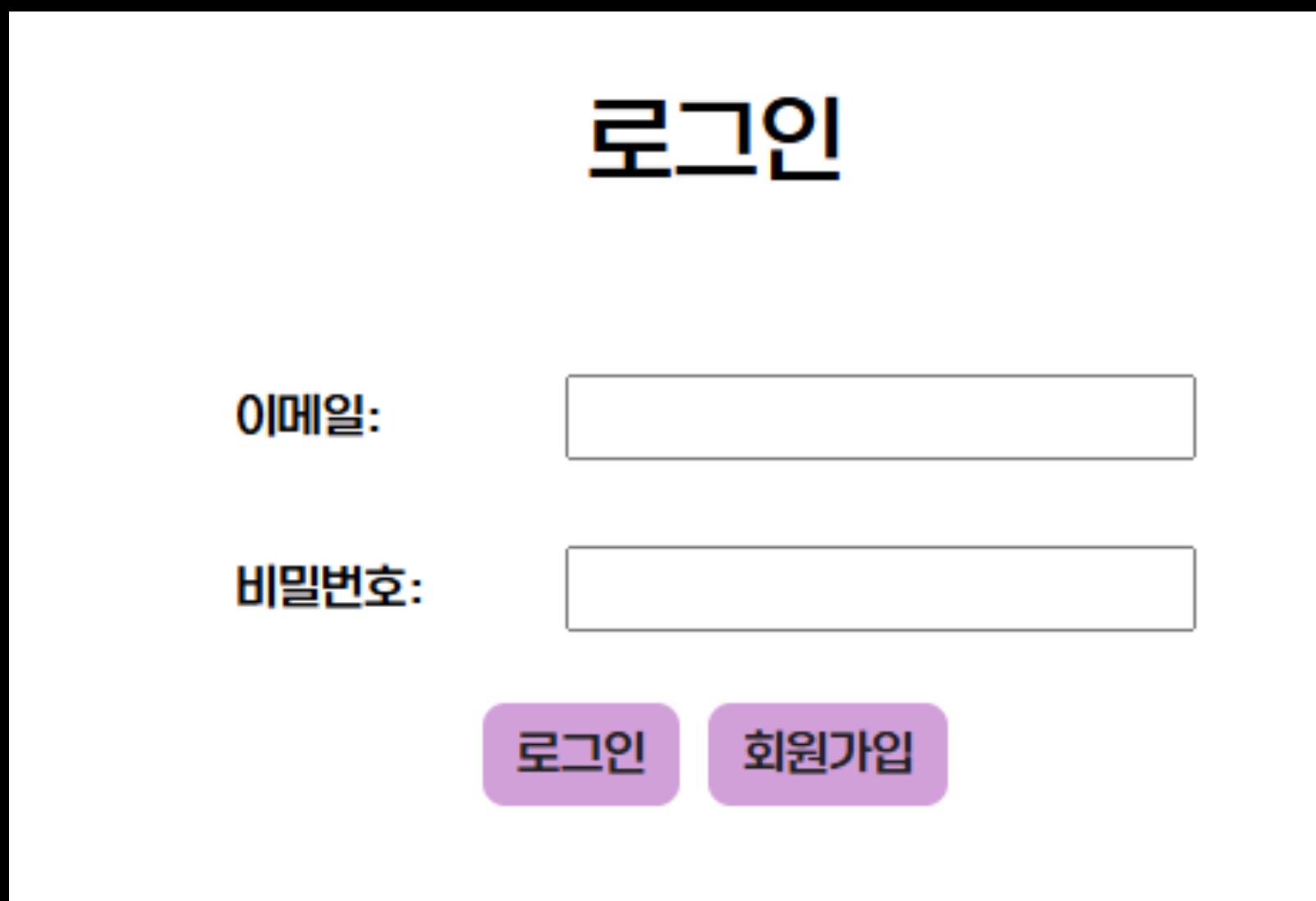
# 웹페이지 열기
driver.get("http://localhost:3000")

# 특정 요소 찾기 및 조작
input_field = driver.find_element("name", "username")
input_field.send_keys("testuser")
login_button = driver.find_element("id", "login-button")
login_button.click()
```

## 실습1 : 로그인 페이지 자동 실행 실습

### 실습 진행 방식

1. Selenium이 Chrome 브라우저를 자동으로 실행하는지 확인
2. 'driver.get("http://localhost:3000/login")'을 통해 로그인 페이지가 정상적으로 열리는지 확인
3. 'print(driver.title)'을 사용하여 페이지 제목이 올바르게 표시되는지 확인
4. 'print(driver.current\_url)'을 활용해 현재 페이지 URL이 '/login'인지 검증



The screenshot shows a login form with the title '로그인' at the top. It has two input fields: '이메일:' and '비밀번호:', each with a corresponding input box. Below the input fields are two buttons: '로그인' and '회원가입'. The '로그인' button is highlighted with a purple background.

### 예상 결과

- 브라우저가 자동으로 실행되고 로그인 페이지가 열려야 함
- 'print(driver.title)' 실행 시 로그인 페이지 제목이 출력됨
- 'print(driver.current\_url)' 실행 시 "http://localhost:3000/login"이 출력됨

```
DevTools listening on ws://127.0.0.1:1311
현재 페이지 제목: React App
현재 URL: http://localhost:3000/login
Enter를 누르면 창이 닫힙니다...Created Te
```



# 비동기 페이지 로딩은 WebDriverWait

## 비동기 요소

- React 웹사이트에서 데이터가 즉시 로드되지 않는 경우, Selenium이 요소를 찾지 못할 수 있음.
- 예) 로그인 버튼이 API 응답 후 활성화될 경우, 버튼을 클릭하려고 하면 오류 발생
- 비동기 로딩 문제 해결 방법 → WebDriverWait 사용!

### ✓ WebDriverWait을 활용한 요소 대기 방법

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

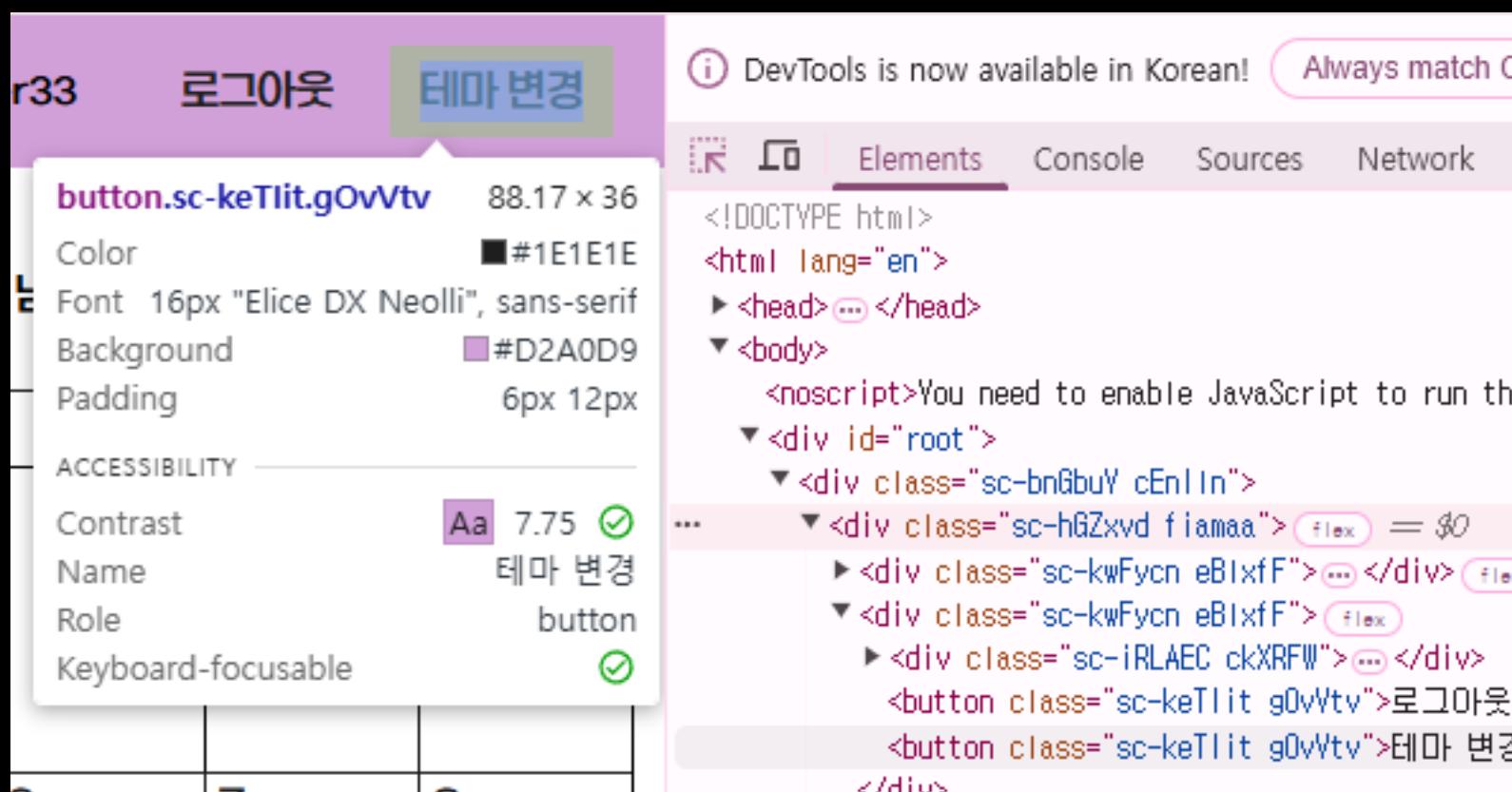
# 로그인 버튼이 활성화될 때까지 최대 10초 대기
wait = WebDriverWait(driver, 10)
login_button = wait.until(EC.element_to_be_clickable((By.ID, "login-button")))
login_button.click()
```



# 개발자 도구(F12)로 웹 요소 쉽게 찾기

## 개발자 도구(F12)

- 웹 브라우저에서 HTML, CSS, JavaScript 코드를 분석할 수 있는 도구
- Selenium이 자동화할 웹 요소를 찾을 때 필수 사용



## 개발자 도구에서 요소 찾는 방법

- Chrome 브라우저에서 F12 키를 눌러 개발자 도구 열기
- 요소 선택기(Inspect, Ctrl+Shift+C) 클릭 후 테스트할 요소 선택
- 요소의 ID, Class, Name 속성 확인
- Selenium에서 사용 가능한 속성으로 선택 방법 결정

```
# ID로 요소 찾기
username_input = driver.find_element("id", "username")

# Name 속성으로 요소 찾기
password_input = driver.find_element("name", "password")

# Class 속성으로 요소 찾기
login_button = driver.find_element("class name", "login-btn")
```



# find\_element vs find\_elements

## ✓ find\_element()와 find\_elements()의 차이점

▶START

```
# 첫 번째로 발견된 로그인 버튼을 찾음  
login_button = driver.find_element("id", "login-button")  
login_button.click()
```

find\_element() : 첫 번째로 발견된 요소만 찾음  
→ 해당 ID를 가진 버튼이 하나만 존재하는 경우 사용하면 좋음

▶START

▶START

▶START

```
# 모든 메뉴 아이템 가져오기  
menu_items = driver.find_elements("class name", "menu-item")  
  
# 첫 번째 메뉴 클릭  
menu_items[0].click()
```

find\_elements() : 모든 일치하는 요소 찾음  
→ 여러 개의 동일한 요소(예: 여러 개의 메뉴 버튼)를 찾고 싶을 때 사용

## execute\_script()

- Selenium에서는 JavaScript를 실행하여 웹 요소를 조작할 수 있음
- 보이지 않는 요소를 강제로 보여주거나, 스크롤을 이동하는 등 다양한 조작 가능
- React 기반 웹사이트처럼 동적 요소가 많을 경우 매우 유용함



### 버튼이 숨겨져 있을 때 강제 클릭하는 방법

```
# 숨겨진 버튼 찾기  
hidden_button = driver.find_element(By.ID, "hidden-button")  
  
# JavaScript로 버튼 강제 클릭  
driver.execute_script("arguments[0].click();", hidden_button)
```



# 테스트 실행 결과를 자동으로 로그 파일에 저장

## 로그(Log)

- 테스트 과정에서 발생한 이벤트(성공/실패/오류 등)를 기록하는 파일
- 자동화 테스트가 정상적으로 수행되었는지 추적할 수 있음
- Selenium 테스트 기록을 남겨 디버깅을 쉽게 할 수 있음

```
# 로그 설정 (파일 저장)
logging.basicConfig(
    filename="selenium_test.log", # 저장할 로그 파일 이름
    level=logging.INFO, # 로그 레벨 (INFO 이상만 저장)
    format"%(asctime)s - %(levelname)s - %(message)s"
)

# WebDriver 실행
driver = webdriver.Chrome()
driver.get("http://localhost:3000/login")

# 로그인 페이지 실행 확인 로그 기록
logging.info("로그인 페이지 실행됨 - URL: %s", driver.current_url)
```

## 로그 파일에 기록하는 방법

1. Python의 `logging` 모듈을 활용하여 로그 생성
2. 로그 레벨(INFO, ERROR 등)을 설정하여 실행 결과를 저장
3. 로그 파일(`selenium_test.log`)에 저장하여 테스트 기록 유지

```
2024-02-24 10:00:01 - INFO - 로그인 페이지 실행됨 - URL: http://localhost:3000/login
2024-02-24 10:00:02 - INFO - 로그인 버튼 발견됨
```

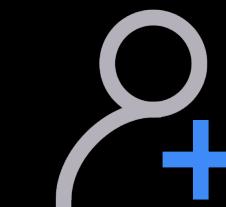


# 회원가입 & 로그인 자동화란?

## 회원가입 & 로그인 자동화의 필요성

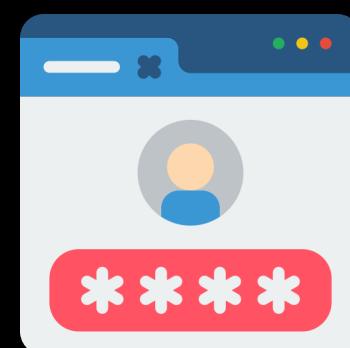
- 반복적인 회원가입/로그인 테스트를 자동화하여 QA 과정 개선
- 폼 입력 및 버튼 클릭을 Selenium으로 자동화 가능

## 회원가입 & 로그인 자동화에서 다룰 핵심 기능



### 1. 회원가입(Signup.tsx)

- 사용자 정보 입력 (find\_element(), send\_keys())
- 회원가입 버튼 클릭 (click())
- 회원가입 성공 후 로그인 페이지로 이동 확인



### 2. 로그인(Login.tsx)

- 기존 계정 정보로 로그인 입력
- 로그인 버튼 클릭 후 UI 변경 확인 (Nav.tsx)
- 로그인 성공 후 로그아웃 버튼 표시 여부 확인



# Selenium으로 로그인 자동화를 하려면?

## 로그인 자동화에서 활용할 Selenium 기능 복습

### 1. 웹 요소 찾기 (`find_element()`)

```
email_input = driver.find_element(By.NAME, "email")
password_input = driver.find_element(By.NAME, "password")
login_button = driver.find_element(By.TAG_NAME, "button")
```

### 2. 입력 및 버튼 클릭 (`send_keys()`, `click()`)

```
email_input.send_keys("testuser@example.com")
password_input.send_keys("password123")
login_button.click()
```

### 3. 로그인 후 UI 변화 확인 (`is_displayed()`)

```
logout_button = driver.find_element(By.XPATH, "//button[text()='로그아웃']")
if logout_button.is_displayed():
    print("✓ 로그인 성공 - 로그아웃 버튼 확인됨!")
```

### 4. 쿠키 활용한 자동 로그인 유지 (`get_cookies()`, `add_cookie()`)

```
cookies = driver.get_cookies() # 현재 로그인 상태에서 저장된 쿠키 가져오기
driver.add_cookie(cookies[0]) # 가져온 쿠키를 활용해 자동 로그인 유지
```



# 회원가입 폼 자동 입력

## 회원가입을 위한 입력 필드 자동 탐색

1. 이름 입력 필드 찾기
2. 이메일 입력 필드 찾기
3. 비밀번호 입력 필드 찾기
4. 비밀번호 확인 필드 찾기

```
username_input = driver.find_element(By.NAME, "username")
email_input = driver.find_element(By.NAME, "email")
password_input = driver.find_element(By.NAME, "password")
password_confirm_input = driver.find_element(By.NAME, "passwordConfirm")
```

## 자동 입력 (send\_keys())

- send\_keys("testuser") → 사용자명 입력
- send\_keys("test@example.com") → 이메일 입력
- send\_keys("password123") → 비밀번호 입력

```
username_input.send_keys("testuser")
email_input.send_keys("test@example.com")
password_input.send_keys("password123")
password_confirm_input.send_keys("password123")
```

## 회원가입 버튼 찾기

- find\_element(By.XPATH, "//button[text()='회원가입']") → 회원가입 버튼 선택

```
signup_button = driver.find_element(By.TAG_NAME, "button") # 첫 번째 버튼 찾기
signup_button.click()
```



# 자동화로 회원가입 완료하기

## 회원가입 버튼 클릭 (click())

- find\_element(By.XPATH, "//button[text()='회원가입']") → 회원가입 버튼 찾기
- click() → 클릭 이벤트 실행

```
signup_button = driver.find_element(By.XPATH, "//button[text()='회원가입']") # 회원가입 버  
튼 찾기  
signup_button.click()
```

## 회원가입 성공 후 페이지 이동 확인

- print(driver.current\_url) → 회원가입 후 페이지 확인
- assert "login" in driver.current\_url → 로그인 페이지로 이동했는지 검증

```
time.sleep(2) # 페이지 로딩 대기  
print("현재 URL:", driver.current_url)  
assert "login" in driver.current_url # URL에 'login'이 포함되었는지 확인
```

## 회원가입 실패 시 오류 메시지 확인

- find\_element(By.CLASS\_NAME, "error-message")
- is\_displayed()로 오류 메시지 표시 여부 확인

```
error_message = driver.find_element(By.CLASS_NAME, "error-message")  
if error_message.is_displayed():  
    print("⚠ 회원가입 오류 발생:", error_message.text)
```

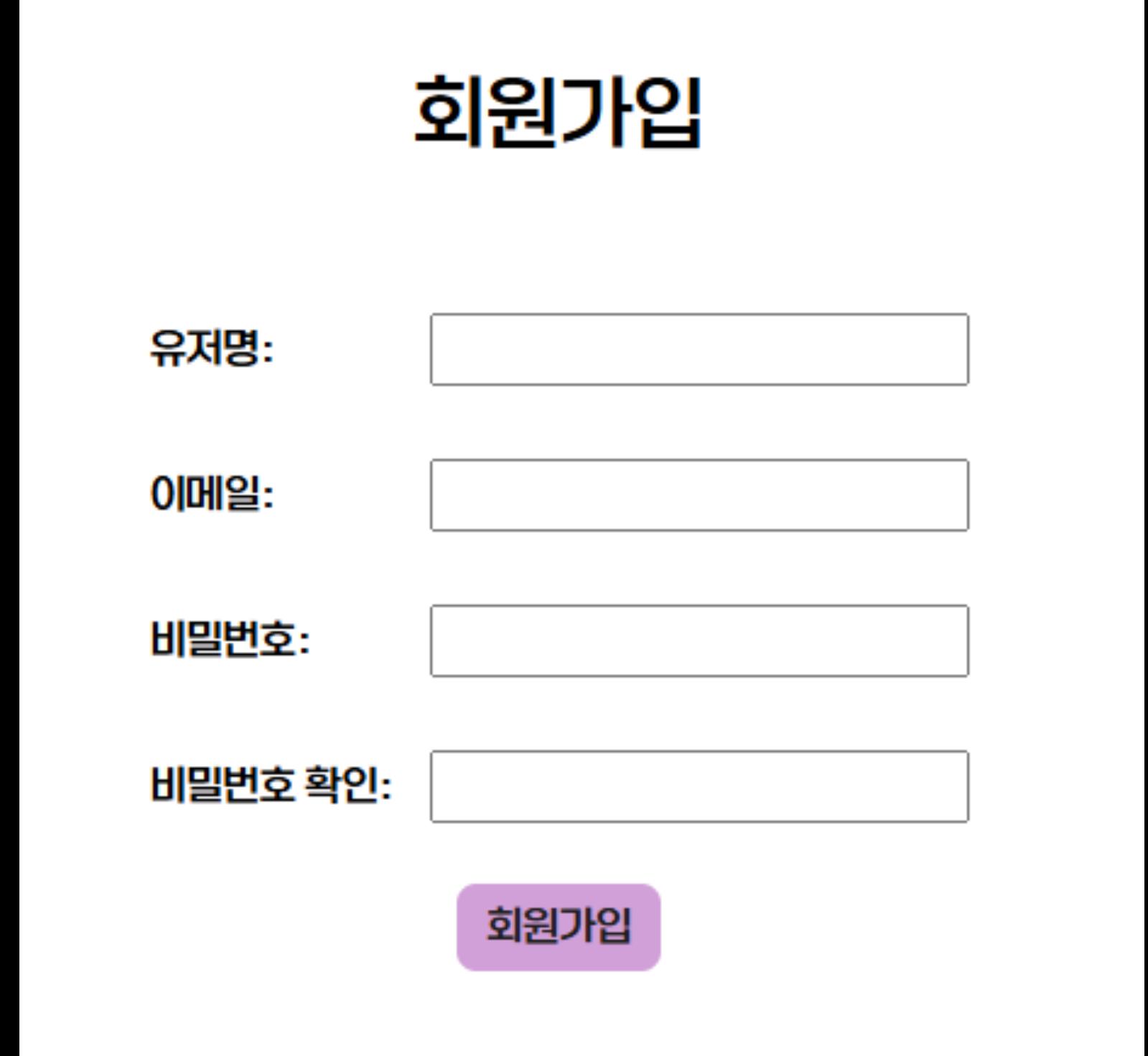
## 실습 2: 회원가입 자동화 테스트 실행

### 실습 목표

- 회원가입 폼에 정보 자동 입력 (send\_keys())
- 회원가입 버튼 자동 클릭 (click())
- 회원가입 성공 Alert 메시지 확인 (switch\_to.alert)
- 회원가입 후 로그인 페이지 이동 확인  
(assert "login" in driver.current\_url)

### 실습 순서

- find\_element(By.NAME, "username") → 사용자명 입력
- find\_element(By.NAME, "email") → 이메일 입력
- find\_element(By.NAME, "password") → 비밀번호 입력
- find\_element(By.NAME, "passwordConfirm")  
→ 비밀번호 확인 입력
- find\_element(By.XPATH, "//button[text()='회원가입']")  
→ 회원가입 버튼 클릭
- driver.switch\_to.alert.accept() → 회원가입 성공 Alert 닫기
- assert "login" in driver.current\_url → 로그인 페이지 이동 확인



회원가입

유저명:

이메일:

비밀번호:

비밀번호 확인:

회원가입



## 회원가입 후 쿠키 확인

### 회원가입 후 쿠키 확인 (get\_cookies())

- driver.get\_cookies() → 현재 브라우저의 쿠키 가져오기
- 회원가입 후에는 로그인 쿠키가 없음 ([] 반환됨)

✖ 회원가입 후 저장된 쿠키: []  
✓ 로그인 페이지로 이동 완료!

### 로그인 후 쿠키 저장 (get\_cookies())

- 로그인 성공 후 get\_cookies() 실행
- 로그인하면 토큰 쿠키가 저장됨

### 자동 로그인 유지 (add\_cookie())

- driver.add\_cookie({...}) → 쿠키를 추가하여 자동 로그인 유지

✖ 로그인 후 저장된 쿠키: [{}{'name': 'token', 'value': 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...'}]  
✓ 이제 로그인 상태가 유지됩니다!



# 로그인 폼 자동 입력

## 로그인 폼 요소 찾기 (find\_element())

- find\_element(By.NAME, "email") → 이메일 입력 필드 찾기
- find\_element(By.NAME, "password") → 비밀번호 입력 필드 찾기
- find\_element(By.XPATH, "//button[text()='로그인']") → 로그인 버튼 찾기

## 로그인 자동 입력 (send\_keys())

- send\_keys("test3@example.com") → 이메일 입력
- send\_keys("password123") → 비밀번호 입력
- click() → 로그인 버튼 클릭

## 로그인 성공 후 URL 확인 (current\_url)

- 로그인 후 'driver.current\_url' 확인하여 '/browse' 페이지로 이동했는지 검증

```
# Chrome WebDriver 실행
driver = webdriver.Chrome()

# 로그인 페이지 이동
driver.get("http://localhost:3000/login")

# ✅ 로그인 폼 요소 찾기
email_input = driver.find_element(By.NAME, "email")
password_input = driver.find_element(By.NAME, "password")
login_button = driver.find_element(By.XPATH, "//button[text()='로그인']")
```



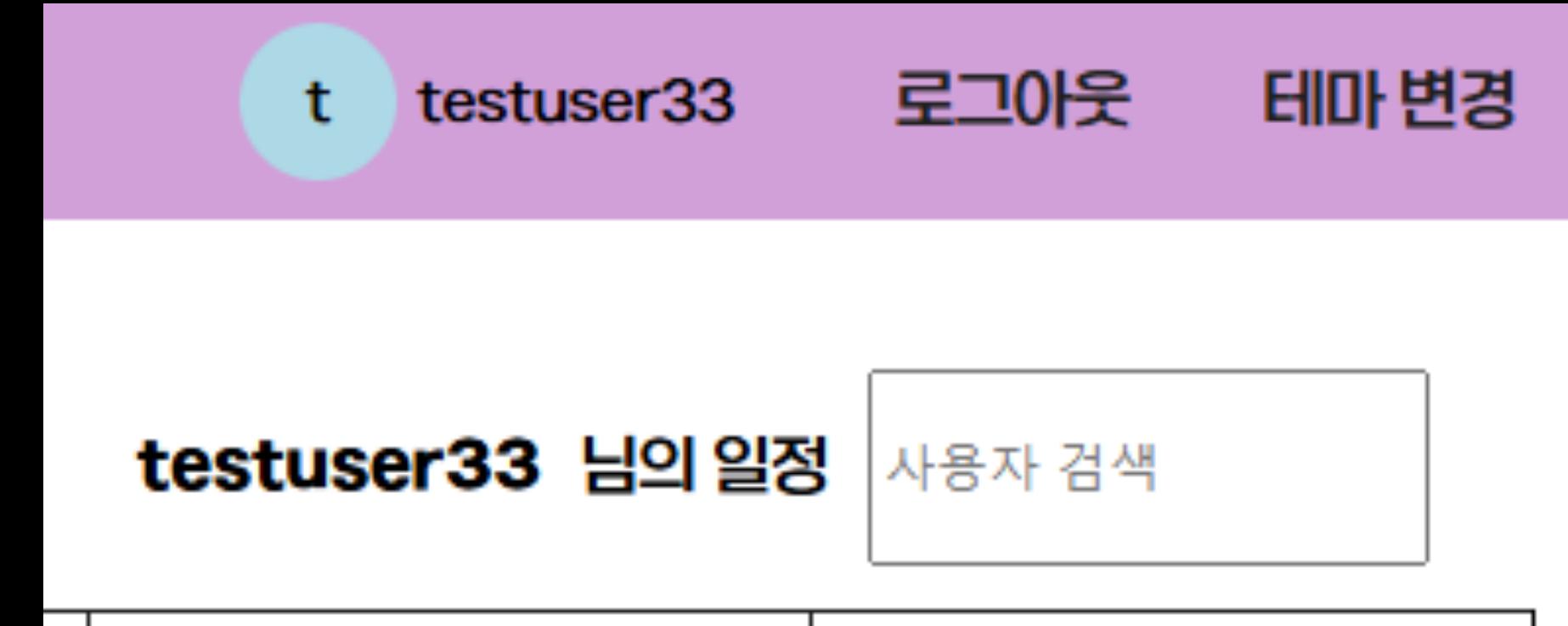
# 로그인 후 UI가 정상적으로 변경되었는지 확인

## 로그인 후 UI 변경 사항 확인하기

- 아바타 버튼이 정상적으로 표시되는지 검사
- 로그아웃 버튼이 제대로 나타나는지 확인
- `is_displayed()` 활용하여 UI 상태 검증

## Selenium으로 UI 변경 검증하기

- `find_element(By.XPATH, "아바타 버튼")`
- `find_element(By.XPATH, "로그아웃 버튼")`
- `assert 요소.is_displayed()`





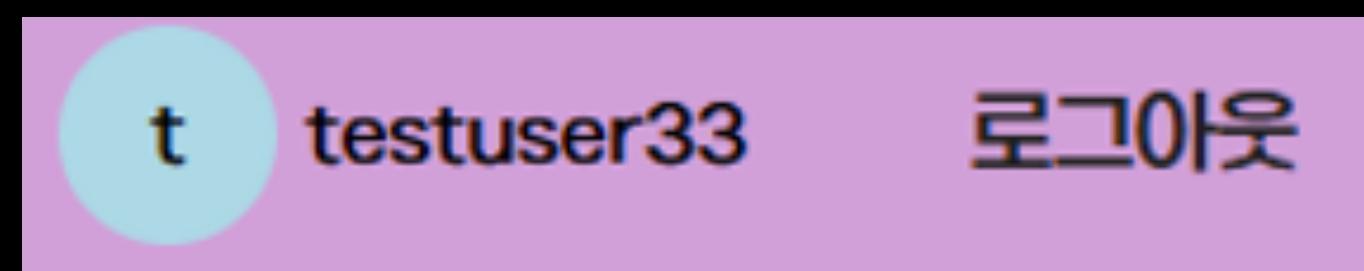
## 실습3 : 로그인 자동화 테스트

### 실습 목표

- Selenium을 활용하여 로그인 자동화 테스트를 구현합니다.
- 로그인 후 URL 변경 및 UI 요소(로그아웃 버튼)를 검증합니다.

### 실습 과정

1. 로그인 자동화 실행: 이메일 & 비밀번호 입력 후 로그인 버튼 클릭
2. 로그인 성공 여부 확인: 현재 URL 변경 확인
3. UI 변경 검증: 로그아웃 버튼이 나타나는지 확인



```
DevTools listening on ws://127.0.0.1:3502/devtools/browser/...
현재 URL: http://localhost:3000/browse
✓ 로그인 후 로그아웃 버튼이 정상적으로 표시되었습니다!
Enter를 누르면 창이 닫힙니다...
```

# ✖ 잘못된 로그인 시 오류 메시지 확인

## 로그인 실패 시 오류 메시지 출력 테스트

- find\_element(By.NAME, "email") → 잘못된 이메일 입력
- find\_element(By.NAME, "password") → 잘못된 비밀번호 입력
- find\_element(By.XPATH, "//button[text()='로그인']") → 로그인 버튼 클릭

## 오류 메시지 확인 (is\_displayed())

- find\_element(By.CLASS\_NAME, "error-message") → 오류 메시지 출력 여부 확인
- assert error\_message.is\_displayed() → 오류 메시지가 나타나는지 검증

```
# ✅ 로그인 버튼 클릭
login_button = driver.find_element(By.XPATH, "//button[text()='로그인']")
login_button.click()

# ✅ 오류 메시지 확인 (로그인 실패 시 UI 변경 확인)
time.sleep(2) # 오류 메시지가 나타날 시간을 확보

try:
    error_message = driver.find_element(By.CLASS_NAME, "error-message")
    assert error_message.is_displayed()
    print("✅ 로그인 실패 시 오류 메시지가 정상적으로 표시됨:", error_message.text)
except Exception as e:
    print("✖ 오류 메시지를 찾을 수 없음.", str(e))
```



## 퀴즈 타임



QUIZ 1번부터 8번을 풀어봐요!



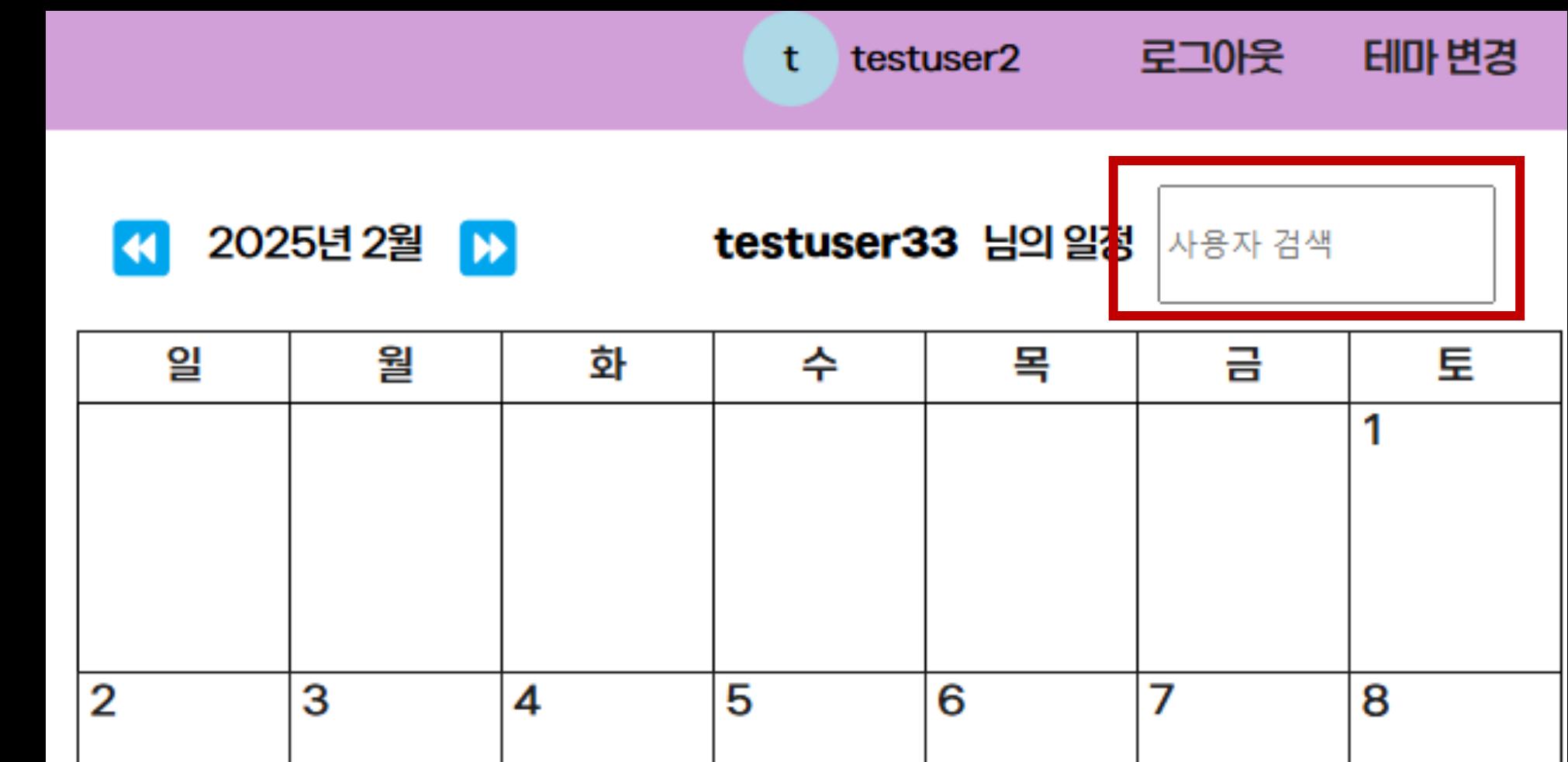
# Selenium으로 사용자 검색 자동화 테스트

## 검색창(SearchUser.tsx)에서 사용자 검색 테스트

- send\_keys() → 검색어 자동 입력
- find\_element(By.XPATH, "//div[text()='사용자이름']") → 검색된 사용자 찾기
- click() → 검색된 사용자 자동 선택

## AJAX 비동기 로딩 대기 (WebDriverWait)

- 검색어 입력 후 검색 결과가 표시될 때까지 대기
- presence\_of\_element\_located() 활용





# send\_keys()로 검색어 입력 & 결과 확인

## 검색창 자동 입력 (send\_keys())

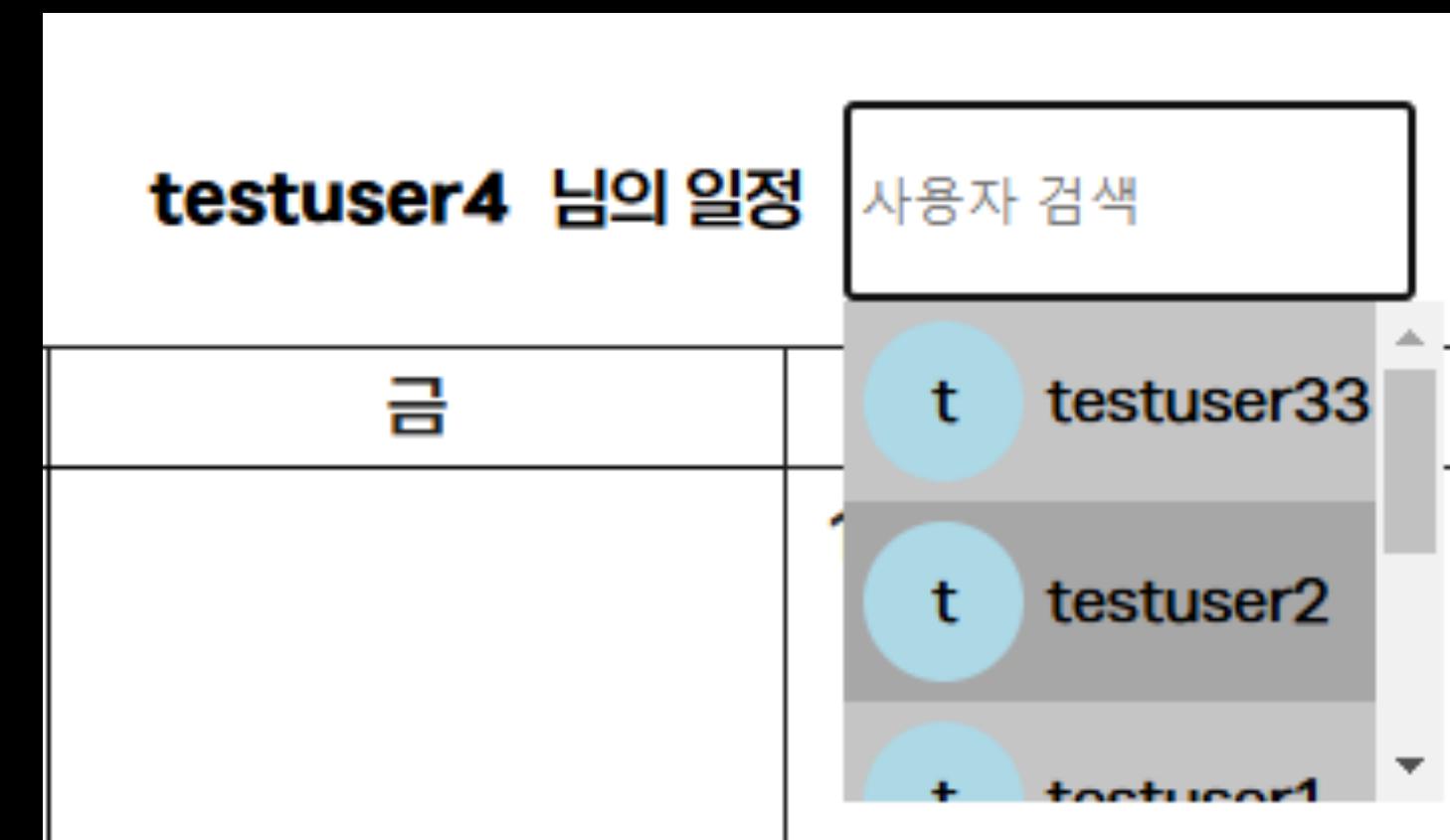
- find\_element(By.CSS\_SELECTOR, "input[placeholder='사용자 검색']") → 검색창 찾기
- send\_keys("testuser") → 자동으로 검색어 입력

## 검색 결과 확인 (find\_elements())

- find\_elements(By.CLASS\_NAME, "avatar-container")  
→ 검색된 사용자 리스트 가져오기
- 검색된 사용자 목록 출력하여 특정 사용자 존재 여부 확인

## 검색된 사용자 클릭 (click())

- find\_element(By.XPATH, "//div[text()='testuser4']")  
→ 검색된 특정 사용자 찾기
- click() → 해당 사용자 선택



# 특정 사용자가 검색 결과에 있는지 검증

## 검색 결과에서 특정 사용자 찾기 (WebDriverWait)

- WebDriverWait(driver, 5).until(EC.presence\_of\_element\_located((By.XPATH, "//div[text()='testuser33']")))
- 검색 결과에서 "testuser33"이 나타날 때까지 대기

## 검색된 사용자 개수 확인

- find\_elements(By.CLASS\_NAME, "avatar-container")  
→ 검색된 모든 사용자 목록 가져오기

```
# ✅ 검색된 모든 사용자 가져오기
users = driver.find_elements(By.XPATH, "//div[text()='testuser33']")

# ✅ 검색된 사용자 수 출력
print(f"◆ 검색된 사용자 수: {len(users)}명")
```

## 검색된 사용자가 특정 사용자인지 확인 (assert)

- assert searched\_user.text == "testuser33"

```
# ✅ 특정 사용자가 검색 결과에 있는지 확인
assert searched_user.text == "testuser33"
print("◆ 'testuser33' 검색 결과 확인 완료!")
```



# 검색 후 필터링 UI 변경

## 검색 후 필터 UI가 변경되었는지 확인 (`is_displayed()`)

- 검색어 입력 후 드롭다운이 표시되는지 검증
- 정확한 XPath를 사용해 동적으로 생성된 요소 찾기

## 필터링 UI 확인 (`is_displayed()`)

- `assert search_dropdown.is_displayed()`
- 검색 후 UI가 정상적으로 나타나는지 체크

## 검색 목록에서 특정 사용자 찾기 (`find_elements()`)

- `find_elements(By.XPATH, "//div[text()='testuser4']")`
- 검색 조건이 올바르게 반영되었는지 확인

```
# ✅ 검색 결과에서 특정 사용자 찾기
searched_user = wait.until(EC.presence_of_element_located(
    (By.XPATH, "//div[text()='testuser4']"))
)

assert searched_user.is_displayed(), "✖️ 검색된 사용자가 보이지 않습니다!"
print("✅ 'testuser4'가 검색 결과에 정상적으로 표시되었습니다.")
```



## 실습4 : Selenium을 활용한 검색 자동화

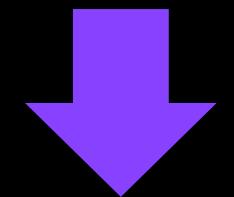
### 실습 목표

- Selenium을 활용하여 검색 기능을 자동화합니다.
- AJAX 기반 검색 결과가 로딩될 때까지 WebDriverWait을 사용해 대기합니다.
- 검색된 사용자가 나타나는지 확인하고 클릭하여 선택합니다.

### 실습 과정

1. 로그인 후 검색창 찾기 → 로그인 후 검색 입력 필드 탐색
2. 검색어 입력 → 특정 사용자명을 입력
3. 검색 결과 대기 (WebDriverWait) → AJAX 로딩 완료까지 대기
4. 검색된 사용자 클릭 → 원하는 검색 결과 선택

testuser33 님의 일정	
수	목



testuser2 님의 일정	
수	목

## AJAX 기반 데이터 로딩

- 검색어 입력 후, 검색 결과가 즉시 표시되지 않고 비동기(AJAX) 요청을 통해 로딩됨
- 페이지 새로고침 없이 데이터가 추가적으로 불러와지는 방식

## WebDriverWait을 활용한 검색 결과 로딩 대기

- WebDriverWait(driver, 5).until(EC.presence\_of\_element\_located(...))
- 특정 요소(검색 결과)가 완전히 로드될 때까지 기다려야 함

```
# ✅ 검색 결과가 나타날 때까지 대기
searched_user = wait.until(EC.presence_of_element_located(
    (By.XPATH, "//input[@placeholder='사용자 검색']/following-sibling::div"))
))
```

## 로딩 완료 후 검색된 사용자 확인 (find\_element())

- 검색어 입력 → AJAX 요청 → 검색 결과 로드 → 사용자 선택
- 검색 결과를 찾을 수 없을 경우 예외 처리 필요



# 원하는 사용자만 정확히 찾는 방법

## 필터 적용 후 UI 변화 감지

- 검색창에 입력한 값에 따라 표시되는 사용자 목록이 변경됨
- 검색 결과에 포함된 사용자만 표시되는지 검증 필요



## assert를 활용한 필터링 검증

- assert "testuser4" in results\_text → 특정 사용자가 포함되어 있는지 확인
- assert "testuser1" not in results\_text → 원하지 않는 사용자가 포함되지 않았는지 확인

## 검색 결과가 필터링되지 않으면?

- 검색 입력 후 AJAX 요청이 정상적으로 실행되었는지 확인
- 기대한 결과가 없을 경우 오류 메시지 출력

```
# ✅ 검색 결과 검증
assert "testuser4" in results_text, "✖ 'testuser4'가 검색 결과에 포함되지 않았습니다!"
print("✅ 'testuser4'가 검색 결과에 올바르게 표시됨.")
```

# 검색 결과 UI, 자동으로 업데이트될까?

## execute\_script()

- JavaScript를 실행하여 UI 업데이트를 강제하는 메서드
- AJAX 요청 후 검색 결과가 자동으로 업데이트되는지 검증 가능

## 검색 결과 UI 업데이트 테스트 방법

- 검색어 입력 후 execute\_script()를 사용해 검색 결과 창을 강제로 업데이트
- UI가 정상적으로 변경되었는지 assert를 활용해 확인

## 검색 결과가 자동 업데이트되지 않는다면?

- execute\_script("return document.querySelectorAll('검색결과 요소')") → 결과 목록 확인
- JavaScript 코드 실행 후 UI가 변경되었는지 확인

```
# ✅ 검색 결과 창 강제 업데이트
driver.execute_script("document.querySelector('input[placeholder=\"사용자 검색\"]').dispatchEvent(new Event('input'))")
print("✅ 검색 결과 UI 강제 업데이트 실행")
```



# 잘못된 검색어 입력 시, '결과 없음' 표시될까?

## 검색어 입력 후 결과 확인

- send\_keys("잘못된검색어") → 존재하지 않는 사용자 입력
- is\_displayed()를 활용해 "결과 없음" 메시지가 표시되는지 확인

## 검색 실패 메시지 검증

- assert "결과 없음" in 메시지 요소.text → 메시지 표시 여부 확인

## UI 변경 사항 체크 (is\_displayed())

- 검색 결과창이 숨겨졌는지 확인 (assert not 검색결과창.is\_displayed())

```
# ✅ "결과 없음" 메시지가 표시될 때까지 대기
no_result_message = wait.until(EC.presence_of_element_located(
    (By.XPATH, "//div[text()='결과 없음']")) # 정확한 메시지 텍스트에 맞춰 조정 필요
))

# ✅ 메시지가 실제로 표시되는지 확인
assert no_result_message.is_displayed(), "✖ '결과 없음' 메시지가 표시되지 않았습니다!"
print("✅ '결과 없음' 메시지가 정상적으로 표시됨.")
```

# 검색된 사용자 클릭 후 상세 페이지 이동

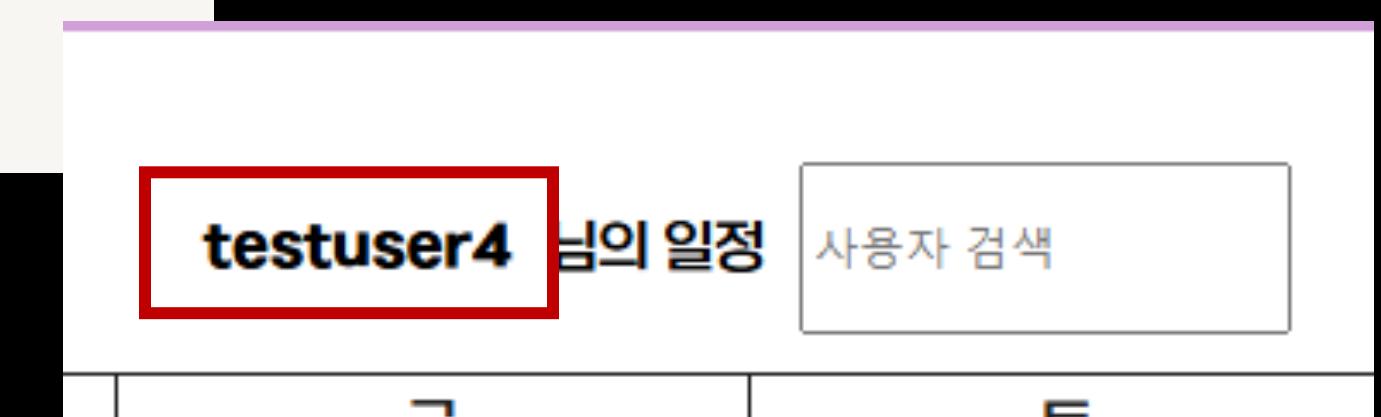
## 사용자 검색 후 클릭 시 페이지 이동 테스트

- find\_element()로 검색된 사용자 찾기
- .click()을 사용해 클릭 이벤트 발생
- 이동한 페이지의 URL 검증 (assert "browse/" in driver.current\_url)

## 이동한 페이지에서 사용자 정보 확인

- find\_element(By.XPATH, "//h1[contains(text(), '님의 일정')]"")
- assert를 활용해 올바른 사용자 페이지인지 확인

```
# ✅ 상세 페이지에서 사용자 이름이 정상적으로 표시되는지 검증
user_name_display = wait.until(EC.presence_of_element_located((By.XPATH, "//h1[contains(t
ext(), '님의 일정')]")))
assert user_name_display.is_displayed(), "❌ 사용자 상세 페이지에서 이름이 표시되지 않았습
니다!"
print("✅ 상세 페이지에서 사용자 이름이 정상적으로 표시됨:", user_name_display.text)
```





# 로그인 상태에 따라 버튼이 바뀌는지 테스트

## 로그인 상태에 따른 UI 변화 확인

- 로그인 전: 로그인 버튼 표시
- 로그인 후: 로그아웃 버튼 및 아바타 표시

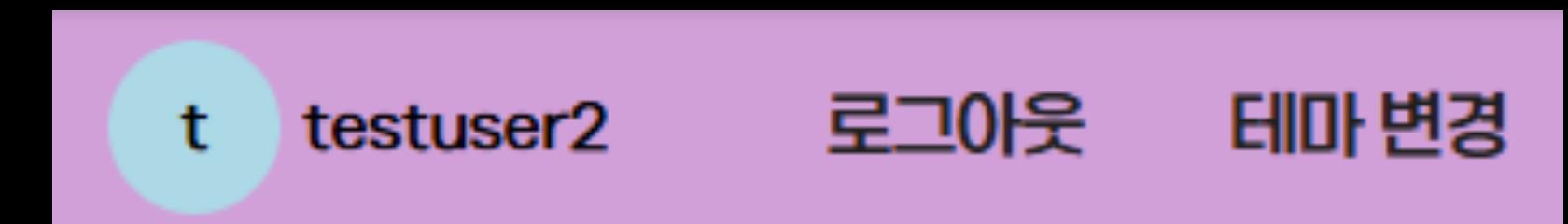
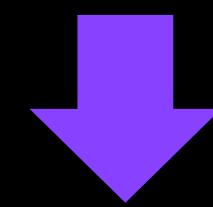
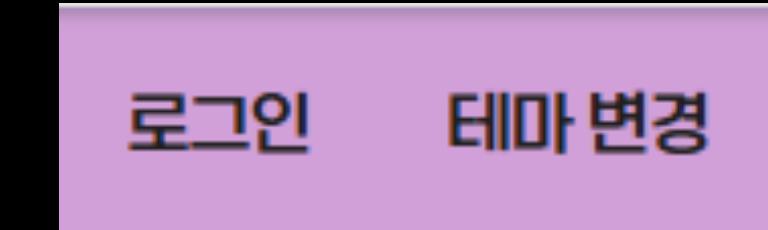
## Selenium을 활용한 테스트 방법

- find\_element(By.XPATH, "//button[text()='로그인']")
- 로그인 후 find\_element(By.XPATH, "//button[text()='로그아웃']")

## UI 변화 검증 (assert is\_displayed())

- assert를 활용해 버튼이 올바르게 표시되는지 확인

```
assert logout_button.is_displayed(), "🔴 로그아웃 버튼이 표시되지 않습니다!"  
assert avatar.is_displayed(), "🔴 아바타가 표시되지 않습니다!"
```





# 버튼을 클릭하면 색상이 변할까?

## 버튼 클릭 후 UI 상태 변경 테스트

- `find_element(By.TAG_NAME, "button").click()` → 버튼 클릭
- `get_attribute("class")`로 스타일 변경 여부 확인

## Selenium으로 버튼 동작 테스트하기

- `is_displayed()`, `is_enabled()`를 활용해 버튼 상태 확인
- `assert`를 사용해 UI 변경 검증

```
# 클릭 전 배경색 저장
before_color = button.value_of_css_property("background-color")

button.click() # 버튼 클릭

# 클릭 후 배경색 다시 가져오기
after_color = button.value_of_css_property("background-color")

assert before_color != after_color, "✗ 버튼 클릭 후 색상이 변하지 않았습니다!"
print("✓ 버튼 클릭 후 색상이 변경됨!")
```



# 입력 필드 자동 입력 & 유효성 검증

## 입력 필드 자동 입력 테스트 (send\_keys())

- find\_element(By.NAME, "email").send\_keys("test@example.com")
- find\_element(By.NAME, "password").send\_keys("password123")

## 유효성 검증 (get\_attribute("value"))

- 입력 후 필드에 값이 정상적으로 반영되었는지 확인
- 필드가 비어 있는지 (assert input\_field.get\_attribute("value") != "")

## 필수 입력 항목 검증 (required 속성 확인)

- get\_attribute("required") == "true" → 필수 입력 여부 확인

```
assert email_input.get_attribute("value") == "test@example.com", "✖ 이메일 입력값이 정상적으로 반영되지 않았습니다!"
assert password_input.get_attribute("value") == "password123", "✖ 비밀번호 입력값이 정상적으로 반영되지 않았습니다!"
print("✓ 입력값이 정상적으로 반영됨!")
```

# Selenium으로 페이지 이동 테스트

## 링크 버튼 자동 클릭 (click())

- find\_element(By.LINK\_TEXT, "일정 열람").click()
- 버튼 클릭 후 페이지가 정상적으로 이동하는지 확인

## 페이지 이동 검증 (assert)

- assert "browse" in driver.current\_url
- 이동 후 URL이 기대한 값과 일치하는지 확인

## 현재 URL 확인 (current\_url)

- print(driver.current\_url) → 현재 URL 출력

```
# ✅ 페이지 이동 후 URL 확인
current_url = driver.current_url
assert "browse" in current_url, "✖️ 페이지 이동이 정상적으로 이루어지지 않았습니다!"
print("✅ 페이지 이동 성공! 현재 URL:", current_url)
```



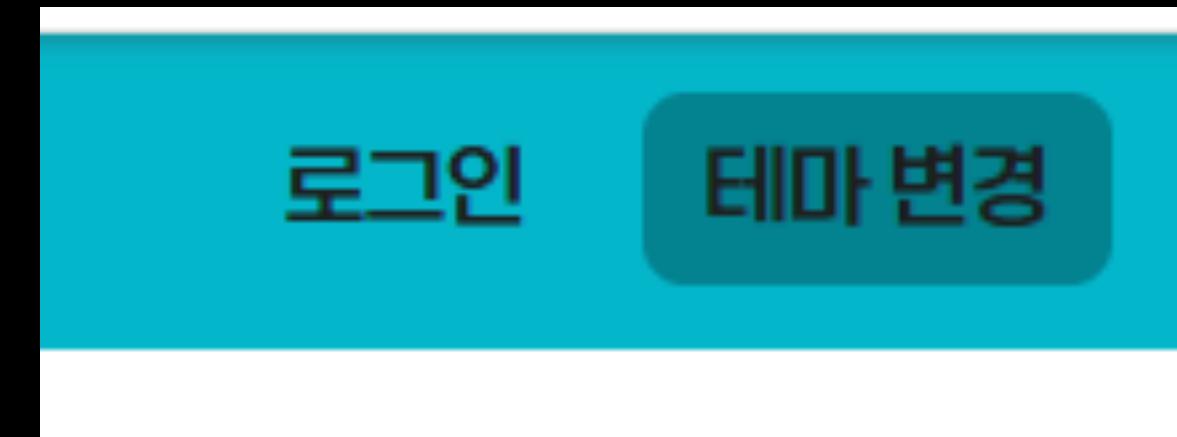
## 실습5 : 테마 변경 버튼 클릭

### 실습 목표

- Selenium을 활용하여 UI 변경 감지 및 검증 자동화
- 테마 변경 버튼 클릭 후 배경색이 변경되는지 확인

### 실습 과정

1. 테마 변경이 적용될 div 요소 찾기
2. 변경 전 배경색 가져오기 (`execute_script()`)
3. 테마 변경 버튼 클릭 (`click()`)
4. 변경된 배경색이 적용될 때까지 대기 (`WebDriverWait`)
5. 변경 후 배경색 가져오기 및 검증 (`assert`)



```
DevTools listening on ws://127.0.0.1:835
● 변경 전 배경색: rgb(210, 160, 217)
● 변경 후 배경색: rgb(4, 182, 202)
✓ 테마 변경 테스트 성공!
Enter를 누르면 창이 닫힙니다...Created T
```



# JavaScript를 활용한 UI 상태 변경 자동화

## Selenium의 `execute_script()` 활용

- `execute_script()` → JavaScript 코드 실행하여 UI 변경
- DOM 조작 (`style.backgroundColor`) → 직접 스타일 변경
- 변경된 상태 확인 (`getComputedStyle()`)

## 테스트 적용 예시

- 버튼 색상 변경 테스트
- 입력 필드 값 변경 테스트
- 페이지 스크롤 조작 테스트

### 페이지 스크롤 조작 테스트

```
# ✅ JavaScript로 페이지 최하단으로 스크롤
driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")

# ✅ 현재 스크롤 위치 확인
scroll_position = driver.execute_script("return window.scrollY;")
print(f"💡 현재 스크롤 위치: {scroll_position}")

# ✅ 스크롤 이동 검증
assert scroll_position > 0, "✖️ 스크롤 조작 실패!"
print("✅ 스크롤 조작 테스트 성공!")
```

# 자동화 테스트에서 변경 전/후 화면 캡처

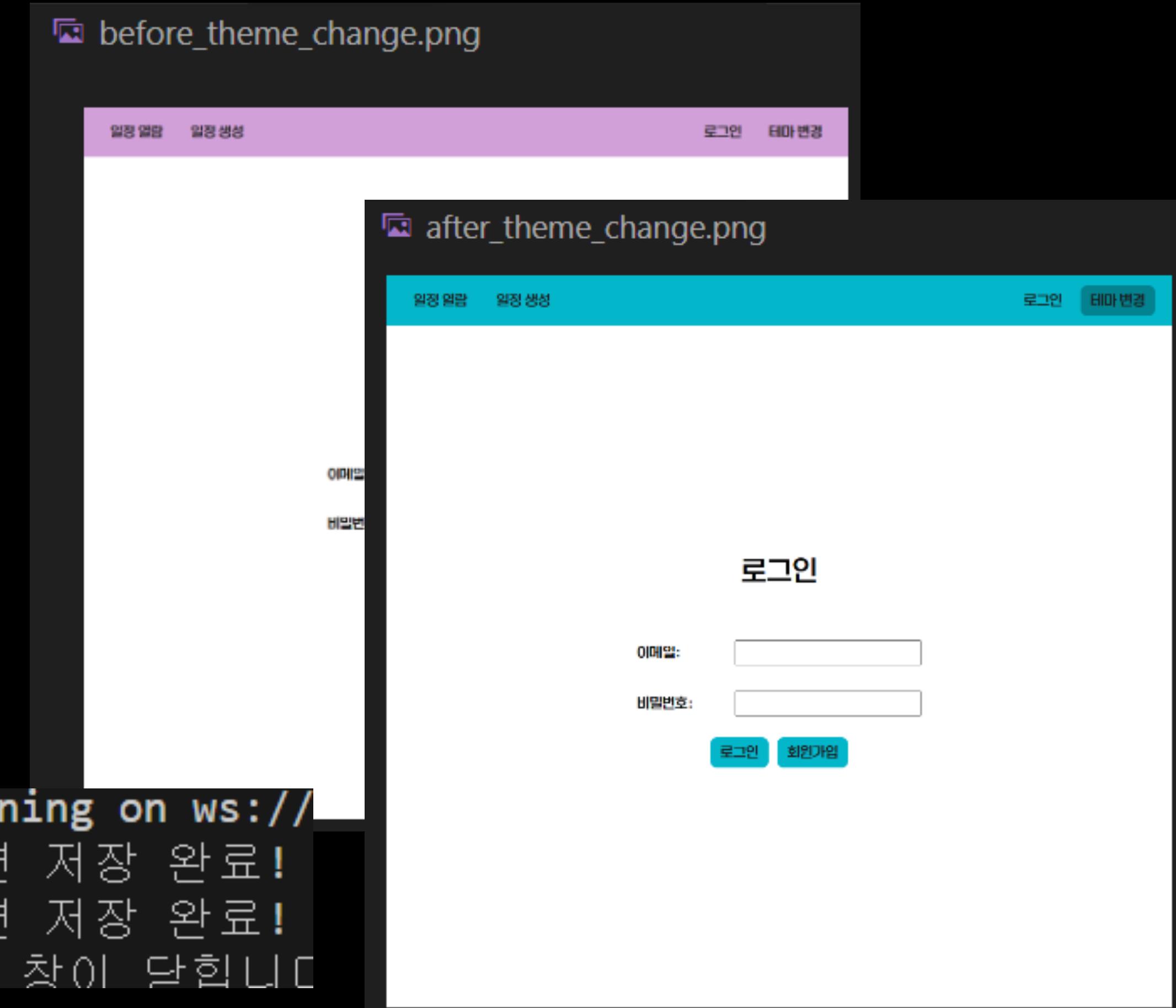
## Selenium의 `save_screenshot()` 활용

- 변경 전 상태 스크린샷 저장
- 테스트 실행 후 변경된 화면 캡처
- 스크린샷 비교하여 UI 변경 여부 확인

## 활용 예시

- 버튼 클릭 전/후 UI 상태 비교
- 테마 변경 후 배경색 비교
- 로그인 전/후 UI 상태 변화 확인

DevTools listening on ws://  
拍照 变更 전 화면 저장 완료!  
拍照 变更 후 화면 저장 완료!  
Enter를 누르면 창이 닫힙니다





# UI 변경 테스트가 실패하는 이유와 해결 방법

## UI 변경 테스트가 실패하는 주요 원인

- 요소 탐색 실패 (NoSuchElementException)
- 비동기 로딩 문제 (ElementNotInteractableException)
- 예상치 못한 UI 렌더링 지연 (TimeoutException)
- 동적 클래스명 변경으로 인한 find\_element() 실패



### 문제

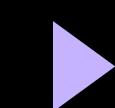
NoSuchElementException: 찾으려는  
요소가 존재하지 않음



### 해결 방안

WebDriverWait을 활용하여 요소가 나타날  
때까지 대기

ElementNotInteractableException: 요소가  
비활성화 상태



is\_enabled()로 요소 상태 확인 후 클릭

getComputedStyle()에서 rgba(0, 0, 0, 0)  
값이 반환됨



WebDriverWait을 활용하여 스타일 변경을  
기다린 후 비교



# UI가 정상적으로 변경되었는지 확인하는 방법

## UI 상태 변경 확인을 위한 주요 메서드

- `is_enabled()` → 버튼이 클릭 가능한지 확인
- `is_displayed()` → 버튼이 화면에 표시되는지 확인
- `get_attribute()` → 버튼의 속성 값을 확인하여 변경 감지

```
# ✅ 버튼 클릭
button.click()

# ✅ "완료" 메시지가 나타날 때까지 대기
success_message = wait.until(EC.presence_of_element_located((By.XPATH, "//div[@id='success-message']")))

# ✅ "완료" 메시지가 정상적으로 표시되었는지 확인
assert success_message.is_displayed(), "✗ 완료 메시지가 표시되지 않았습니다."
print("✅ 버튼 클릭 후 UI 변경 확인 완료!")
```

## 버튼 클릭 후 UI 반영 테스트 시나리오

1. 버튼이 클릭 가능한 상태인지 확인 → `is_enabled()`
2. 버튼 클릭 후 상태 변경을 기다림 → `WebDriverWait`
3. 버튼이 비활성화되었는지 확인 (`disabled` 속성 체크)
4. 버튼 클릭 후 새로운 UI 요소가 표시되는지 검증 (`is_displayed()`)

# 데이터가 반영되었는지 확인하는 방법

## UI 변경 후 데이터 반영 여부를 확인하는 주요 메서드

- find\_element() → 특정 UI 요소를 찾아 데이터 값 확인
- get\_attribute("value") → 입력 필드 값이 변경되었는지 확인
- text 속성 활용 → 버튼/라벨 등의 텍스트 변경 감지

## 데이터 반영 테스트 시나리오

1. UI 요소를 찾기 (find\_element())
2. 버튼 클릭 또는 입력값 변경
3. 데이터가 정상적으로 업데이트되었는지 검증 (get\_attribute(), text)

```
# ✅ 버튼 클릭 후 상태 확인
print(f"🔍 변경 후 버튼 상태: {submit_button.get_attribute('disabled')}")


# ✅ 데이터 반영 검증
assert submit_button.get_attribute("disabled") == "true", "❌ 버튼이 비활성화되지 않았습니다."
print("✅ 버튼 상태 변경 확인 완료!")
```



## 퀴즈 타임



QUIZ 9번부터 16번을 풀어봐요!