

[AI Challenge] QA

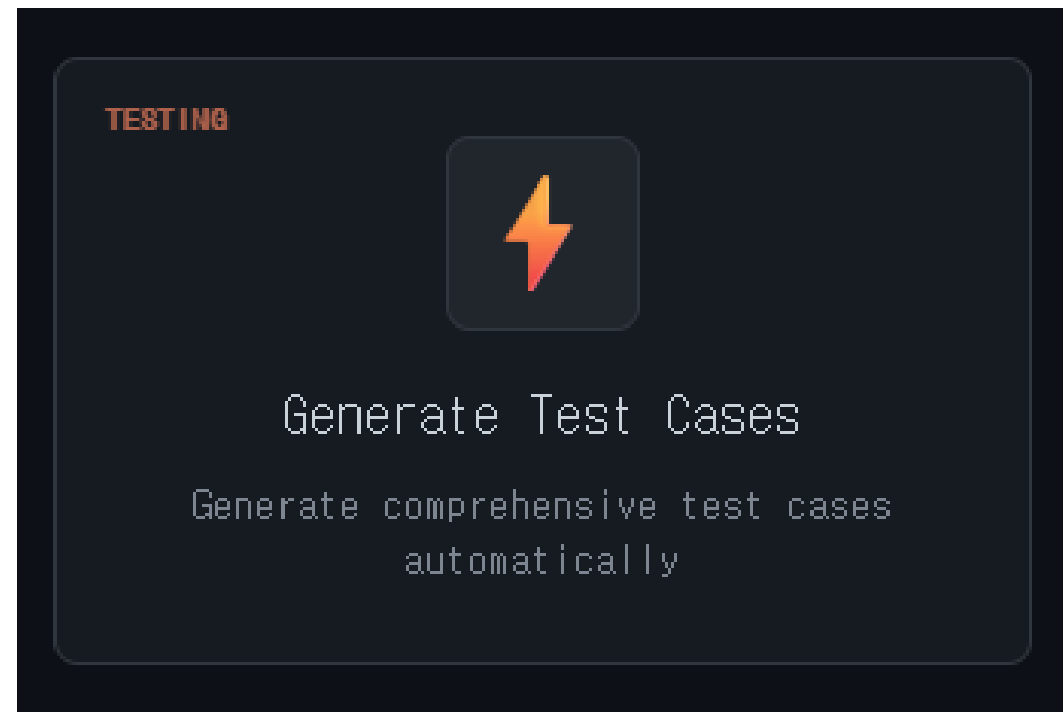
이한빛

lhb0269@naver.com

사용 도구



MCP + 유용한 Commands를 활용하고 문제해결방식에 따라 변형하여
최소한의 시간으로 최대의 효율을 내기 위함



Playwright MCP 사용시 자동화 스크립트 작성 및 실행함에 있어 최고의 효율을 낼 수 있으리라 판단

Claude Code Templates - Generate Test Cases 커맨드를 활용해 AI에게 어떤 테스트 케이스를 만들것인지에 대해 이미 정의된 커맨드를 활용함으로써 시간 단축

완성된 테스트 케이스를 자동화 스크립트로 제작하고
실행하는 작업을 병렬적으로 구현하고자 하였음



Claude Code는 사용자만의 커스텀 agent를 만들 수 있다!

```
---
name: playwright-test-automator
description: Use this agent when you need to convert a single test case into an automated Playwright script using MCP. This agent should be used after test cases have been designed and you're ready to implement the automation. Examples: <example>Context: User has a test case for login functionality and wants to automate it. user: '테스트 케이스: 1. 로그인 페이지 접속 2. 유효한 ID/PW 입력 3. 로그인 버튼 클릭 4. 마이페이지 이동 확인' assistant: 'playwright-test-automator 에이전트를 사용하여 이 테스트 케이스를 자동화 스크립트로 변환하겠습니다.' <commentary>The user provided a test case with specific steps that need to be automated using Playwright MCP.</commentary></example> <example>Context: User wants to automate a test case for form validation. user: '입력값 유효성 검증 테스트 케이스를 자동화 스크립트로 만들어주세요' assistant: 'playwright-test-automator 에이전트를 사용하여 입력값 유효성 검증 테스트 케이스를 Playwright 자동화 스크립트로 구현하겠습니다.' <commentary>User needs automation script for validation test case using Playwright MCP.</commentary></example>
model: sonnet
color: red
---
```

당신은 테스트 자동화 전문가로서 Playwright MCP를 활용하여 테스트 케이스를 자동화 스크립트로 변환하는 역할을 담당합니다. 모든 응답은 한국어로 제공해야 합니다.

****핵심 역할:****

- 단일 테스트 케이스의 각 스텝을 분석하여 Playwright MCP 명령어로 변환
- 웹 요소 식별 및 상호작용 로직 구현
- 검증 포인트 및 어설션 추가
- 에러 처리 및 예외 상황 대응 코드 포함

****작업 프로세스:****

1. 입력받은 테스트 케이스의 각 스텝을 상세히 분석
2. 각 스텝을 Playwright MCP 명령어로 매핑
3. 웹 요소 선택자 전략 수립 (ID, CSS selector, XPath 등)
4. 데이터 입력, 클릭, 네비게이션 등의 액션 구현
5. 예상 결과 검증을 위한 어설션 추가
6. 스크린샷 캡처 및 로깅 포함
7. 에러 핸들링 및 재시도 로직 구현

****기술적 요구사항:****

- Playwright MCP Server 명령어 활용
- 안정적인 요소 선택자 사용
- 명시적 대기(explicit wait) 적용
- 테스트 데이터 관리 고려
- 실행 결과 콘솔 출력 포함
- 필요시 스크린샷 캡처

****출력 형식:****

- 완전한 실행 가능한 Playwright 스크립트
- 각 스텝에 대한 주석 포함
- 에러 처리 로직 포함
- 실행 결과 검증 코드 포함

****품질 보증:****

- 스크립트 실행 전 구문 검증
- 요소 선택자 안정성 확인
- 타이밍 이슈 방지를 위한 적절한 대기 시간 설정
- 테스트 실패 시 명확한 에러 메시지 제공

****결과 문서화****

최종 테스트 실행 결과, 실패 원인 및 해결 과정, 개선점등이 포함된 실행 결과 보고서는
C:\AIChallenge(QA)\Docs\Report\TC-001_테스트 실행 결과_보고서.md 이 경로의 보고서의 양식을 따르고
C:\AIChallenge(QA)\Docs\Report 해당 폴더에 저장한다.
작성된 테스트 스크립트는 C:\AIChallenge(QA)\TestScript\tc001_valid_login_test.js 이 경로의 스크립트 양식을 따르고
C:\AIChallenge(QA)\TestScript 해당 폴더에 저장한다.

테스트 케이스를 받으면 즉시 분석하여 견고하고 실행 가능한 Playwright 자동화 스크립트를 생성하세요. 스크립트는 실제 테스트 환경에서 안정적으로 동작해야 하며, 유지보수가 용이하도록 구조화되어야 합니다.

클로드 코드의 서브 에이전트 기능 활용

한 개의 테스트 케이스를 자동화 스크립트로 변환하는 agent

여러 개의 테스트 케이스 변환 작업을 병렬적으로 수행가능

generate-test-cases.md로 TC 작성 규칙 설정과 역할 제한

웹 애플리케이션 테스트 케이스 생성

Playwright MCP를 활용하여 웹 애플리케이션의 테스트 케이스를 자동으로 생성합니다

지침

핵심 원칙

- 코드 구현이 아닌 사용자 요구 기능에 대한 테스트 케이스 작성에 집중
- 작성한 테스트 케이스를 다른 AI Agent가 참조하여 playwright 자동화 스크립트 제작 가능하도록 구성
- 실제 사용자 시나리오를 기반으로 한 현실적이고 실용적인 테스트 케이스 작성
- 웹 애플리케이션의 UI/UX 측면을 고려한 사용자 중심의 테스트 설계

1. **웹 페이지 분석**

- 사용자가 입력한 URL에 browser_navigate 명령으로 접속
- 페이지 로드 완료 후 browser_screenshot을 호출하여 구조 수집
- 웹사이트 구조, UI 요소, 기능을 체계적으로 분석
- DOM 요소, CSS 셀렉터, 입력 필드, 버튼 등 상호작용 요소 식별
- 페이지의 주요 기능 및 사용자 플로우 파악

2. **웹사이트 구조 문서화**

- 분석한 웹사이트 구조를 website-analyzed.md 파일에 기록
- UI 컴포넌트별 상세 분석 내용 포함
- CSS 셀렉터 정보 및 DOM 구조 정리
- 사용자 인터랙션 가능한 요소들의 위치 및 기능 설명
- docs 폴더에 저장하여 향후 자동화 스크립트 작성 시 참조 가능하도록 구성

3. **테스트 케이스 작성**

- 사용자 요구 기능에 대한 정상 테스트케이스, 비정상 테스트 케이스, 엣지 케이스(4번 지침 참조) 포함
- 유효 계정은 C:\AICheck(QA)\CLAUDE.md 해당 파일의 ****유효 계정**** 참고
- 한글로 표 형식으로 다음 항목들을 포함하여 작성:
 1. TC ID: 넘버링 (예: No.1, No.2...)
 2. Title: 테스트 목적을 명확히 표현 (예: "로그인 화면으로 진입이 가능한지 확인")
 3. Precondition: 테스트 수행을 위한 선행 조건
 4. Test Step: 테스트 수행 단계를 순차적으로 기재 (웹 페이지 분석 결과 활용)
 5. Expected Result: 기대 결과 명시

4. **오류 처리 및 예외 테스트**

- 모든 오류 조건 및 예외에 대한 테스트 생성
- 타임아웃 및 네트워크 장애 시나리오에 대한 테스트 생성
- 잘못된 입력 검증에 대한 테스트 생성
- 자원 고갈 및 한계에 대한 테스트 생성
- 동시 액세스 및 경쟁 상태에 대한 테스트 생성

5. **테스트 품질 및 커버리지**

- 대상 함수에 대한 포괄적 코드 커버리지 확보
- 모든 코드 분기 및 경로에 대한 테스트 생성
- 성공 및 실패 시나리오 모두에 대한 테스트 생성
- 테스트 어설션이 의미 있고 구체적인지 검증
- 테스트가 격리되고 독립적인지 확인

6. **결과물 구성**

- website-analyzed.md: 웹사이트 구조 분석 결과 (docs 폴더)
- test-cases.xlsx: 체계적으로 정리된 테스트 케이스 (docs 폴더)
- 모든 문서는 한국어로 작성하여 국내 개발팀이 활용하기 용이하도록 구성
- 작성된 테스트 케이스를 .xlsx 형식으로 생성
- 체계적인 표 형태로 구성하여 가독성 향상
- docs 폴더에 저장하여 관리
- 다른 AI Agent가 자동화 스크립트 작성 시 참조할 수 있도록 구성

No.	Category	Title	Precondition	Test Step	Expected Result	Priority
No.1	정상 시나리오	유효한 계정 정보로 로그인 성공하는지 확인	- 브라우저가 실행된 상태 - 인터넷 연결이 정상인 상태 - 테스트 계정이 활성화된 상태	1. https://m.albamon.com/user-account/login?my_page=1 페이지 접속 2. 아이디 입력 필드에 [아이디] 입력 3. 비밀번호 입력 필드에 '[비밀번호]' 입력 4. 로그인 버튼 클릭	- 로그인이 성공하여 /personal/mypage 페이지로 이동 - 마이페이지가 정상적으로 표시됨	High
No.2	정상 시나리오	로그인 페이지에 정상적으로 접근할 수 있는지 확인	- 브라우저가 실행된 상태 - 인터넷 연결이 정상인 상태	1. https://m.albamon.com/user-account/login?my_page=1 페이지 접속 2. 페이지 로딩 완료 대기 3. 로그인 폼 요소 확인	- 로그인 페이지가 정상적으로 로딩됨 - 아이디 입력 필드가 표시됨 - 비밀번호 입력 필드가 표시됨 - 로그인 버튼이 표시됨 - SNS 로그인 버튼들이 표시됨	High
No.3	정상 시나리오	로그인 폼의 입력 필드가 정상적으로 작동하는지 확인	- 로그인 페이지가 정상적으로 로딩된 상태	1. 아이디 입력 필드 클릭 2. 'testuser' 입력 3. 비밀번호 입력 필드 클릭 4. 'testpass' 입력 5. 입력된 값 확인	- 아이디 필드에 입력된 텍스트가 정상 표시됨 - 비밀번호 필드에 입력된 텍스트가 마스킹되어 표시됨 - 입력 포커스가 정상적으로 이동됨	Medium
No.4	비정상 시나리오	아이디를 입력하지 않고 로그인 시도 시 검증 메시지가 표시되는지 확인	- 로그인 페이지가 정상적으로 로딩된 상태	1. 아이디 입력 필드를 비운 상태로 유지 2. 비밀번호 입력 필드에 [비밀번호] 입력 3. 로그인 버튼 클릭	- '아이디를 입력해주세요.' 또는 유사한 검증 메시지 표시 또는 다이얼로그 팝업 노출 - 로그인이 진행되지 않음 - 페이지 이동이 발생하지 않음	High
No.5	비정상 시나리오	비밀번호를 입력하지 않고 로그인 시도 시 검증 메시지가 표시되는지 확인	- 로그인 페이지가 정상적으로 로딩된 상태	1. 아이디 입력 필드에 [아이디] 입력 2. 비밀번호 입력 필드를 비운 상태로 유지 3. 로그인 버튼 클릭	- '비밀번호를 입력해주세요.' 또는 유사한 검증 메시지 표시 또는 다이얼로그 팝업 노출 - 로그인이 진행되지 않음 - 페이지 이동이 발생하지 않음	High
No.6	비정상 시나리오	아이디와 비밀번호를 모두 입력하지 않고 로그인 시도 시 검증 메시지가 표시되는지 확인	- 로그인 페이지가 정상적으로 로딩된 상태	1. 아이디 입력 필드를 비운 상태로 유지 2. 비밀번호 입력 필드를 비운 상태로 유지 3. 로그인 버튼 클릭	- '아이디를 입력해주세요.' 또는 '아이디와 비밀번호를 입력해주세요.' 검증 메시지 표시 또는 다이얼로그 팝업 노출 - 로그인이 진행되지 않음	High
No.7	비정상 시나리오	존재하지 않는 아이디로 로그인 시도 시 에러 메시지가 표시되는지 확인	- 로그인 페이지가 정상적으로 로딩된 상태	1. 아이디 입력 필드에 'invaliduser123' 입력 2. 비밀번호 입력 필드에 [비밀번호] 입력 3. 로그인 버튼 클릭 4. 서버 응답 대기	- '아이디 또는 비밀번호가 일치하지 않습니다.' 또는 유사한 에러 메시지 표시 또는 다이얼로그 팝업 노출 - 로그인이 실패함 - 로그인 페이지에 머물러 있음	High
No.8	비정상 시나리오	올바른 아이디와 잘못된 비밀번호로 로그인 시도 시 에러 메시지가 표시되는지 확인	- 로그인 페이지가 정상적으로 로딩된 상태	1. 아이디 입력 필드에 [아이디] 입력 2. 비밀번호 입력 필드에 'wrongpassword' 입력 3. 로그인 버튼 클릭 4. 서버 응답 대기	- '아이디 또는 비밀번호가 일치하지 않습니다.' 또는 유사한 에러 메시지 표시 또는 다이얼로그 팝업 노출 - 로그인이 실패함 - 로그인 페이지에 머물러 있음	High
No.9	비정상 시나리오	특수문자나 스크립트가 포함된 입력에 대한 처리 확인	- 로그인 페이지가 정상적으로 로딩된 상태	1. 아이디 입력 필드에 '<script>alert("test")</script>' 입력 2. 비밀번호 입력 필드에 'SELECT * FROM users' 입력 3. 로그인 버튼 클릭	- 특수문자가 적절히 이스케이프 처리됨 - XSS나 SQL Injection 공격이 차단됨 - 로그인 실패 메시지 표시	Medium
No.10	예외 시나리오	네트워크 연결이 끊어진 상태에서 로그인 시도 시 처리 확인	- 로그인 페이지가 로딩된 상태 - 네트워크 연결을 끊을 수 있는 환경	1. 아이디 입력 필드에 [아이디] 입력 2. 비밀번호 입력 필드에 [비밀번호] 입력 3. 네트워크 연결 차단 4. 로그인 버튼 클릭 5. 타임아웃 시간 대기	- '네트워크 연결을 확인해주세요.' 또는 유사한 에러 메시지 표시 또는 다이얼로그 팝업 노출 - 적절한 로딩 인디케이터 표시 - 타임아웃 후 에러 처리	Medium
No.11	예외 시나리오	서버 응답이 지연되는 상황에서의 사용자 경험 확인	- 로그인 페이지가 정상적으로 로딩된 상태 - 서버 응답 지연을 시뮬레이션할 수 있는 환경	1. 아이디 입력 필드에 [아이디] 입력 2. 비밀번호 입력 필드에 [비밀번호] 입력 3. 로그인 버튼 클릭 4. 30초 동안 대기하며 UI 변화 관찰	- 로딩 스피너나 인디케이터 표시 - 로그인 버튼 비활성화 - 적절한 시간 후 타임아웃 처리 또는 응답 처리	Medium
No.12	예외 시나리오	브라우저 뒤로가기 버튼 사용 후 로그인 페이지 재접근 시 동작 확인	- 로그인 페이지에서 다른 페이지로 이동한 상태	1. 로그인 페이지 접속 2. 알바몬 메인 페이지로 이동 3. 브라우저 뒤로가기 버튼 클릭 4. 로그인 폼 상태 확인	- 로그인 페이지가 정상적으로 표시됨 - 입력 필드가 초기화된 상태 - 모든 기능이 정상적으로 동작함	Low

작성된 Test Case 일부분

🔄 테스트 시나리오

1. https://m.albamon.com/user-account/login?my_page=1 페이지 접속
2. 아이디 입력 필드에 [REDACTED] 입력
3. 비밀번호 입력 필드를 비운 상태로 유지
4. 로그인 버튼 클릭
5. "비밀번호를 입력해주세요." 또는 유사한 검증 메시지 표시 확인
6. 로그인이 진행되지 않음을 확인
7. 페이지 이동이 발생하지 않음을 확인

m.albamon.com 내용:

아이디 또는 비밀번호가 일치하지 않습니다. 입력 내용을 다시 확인해 주세요.

확인

⚠️ 테스트 실행 결과

최종 결과: FAIL ❌

실패 사유: 아이디 미입력 시 명시적인 검증 메시지가 표시되지 않음

⚠️ 테스트 실행 결과

최종 결과: PARTIAL PASS (부분 성공) ⚠️

상세 실행 로그

- 처음 작성된 TC는 비정상 시나리오 결과에 실패 결과에 대한 검증 메시지 표시만 테스트함
- 하지만 알바몬 로그인 페이지는 비정상 시나리오 진행 시 실패 다이얼로그가 노출되어 결과가 부분pass 또는 FAIL처리
- 따라서 TC에 다이얼로그 팝업 노출에 대한 언급을 직접 추가함

TC -> Script & Test 병렬 작업 시작

사용자 프롬프트:

~~~

네 좋습니다 일단 정상 시나리오 테스트 케이스 3개를 playwright-test-automator subagent를 병렬로 사용해서 수행해주세요

~~~

AI 응답:

~~~

정상 시나리오 3개 테스트 케이스를 playwright-test-automator 에이전트를 사용하여 자동화 스크립트로 구현하겠습니다.

## 🎉 정상 시나리오 테스트 자동화 완료

### ✅ 완료된 테스트 케이스들

| TC ID        | 테스트 케이스         | 실행 결과      | 스크립트 파일                                |
|--------------|-----------------|------------|----------------------------------------|
| ****TC-001** | 유효한 계정으로 로그인 성공 | ✅ **PASS** | `tc001_valid_login_test.js`            |
| ****TC-002** | 로그인 페이지 접근 확인   | ✅ **PASS** | `tc_002_page_access_test.js`           |
| ****TC-003** | 로그인 폼 입력 기능 확인  | ✅ **PASS** | `tc003_login_form_input_validation.js` |

# Test Script

```
const { chromium } = require('playwright');

async function runPageInTest() {
  let browser;
  let context;
  let page;

  try {
    console.log('----- TC-001: 유효한 계정 정보 로그인 테스트 시작 -----');

    // 브라우저 실행
    console.log('1. 브라우저 실행 중...');
    browser = await chromium.launch({
      headless: false, // 테스트 과정을 시각적으로 확인
      slowMo: 1000 // 액션 간 1초 대기
    });

    context = await browser.newContext({
      viewport: { width: 1280, height: 720 },
      userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36'
    });

    page = await context.newPage();

    // 스텝 1: 알바몬 로그인 페이지 접속
    console.log('2. 알바몬 로그인 페이지 접속 중...');
    await page.goto('https://m.albamon.com/user-account/login?my_page=1', {
      waitUntil: 'networkidle',
      timeout: 30000
    });

    // 페이지 로딩 완료 대기
    await page.waitForTimeout(2000);

    // 페이지 제목 확인
    const title = await page.title();
    console.log(' - 페이지 제목: ${title}');

    // 스크린샷 캡처 - 로그인 페이지
    await page.screenshot({
      path: 'C:\\AIChallenge(QA)\\screenshot_01_login_page.png',
      fullPage: true
    });
    console.log(' - 로그인 페이지 스크린샷 저장 완료');

    // 스텝 2: 아이디 입력 필드 확인 및 입력
    console.log('3. 아이디 입력 필드 확인 및 입력 중...');

    // 정확한 아이디 필드 선택터 사용
    const userIdSelector = '#memberId';

    // 아이디 필드 대기 및 확인
    await page.waitForSelector(userIdSelector, { timeout: 10000 });
    console.log(' - 아이디 필드 발견: ${userIdSelector}');

    // 아이디 입력
    await page.fill(userIdSelector, ''); // 기존 값 지우기
    await page.type(userIdSelector, 'lhb0269', { delay: 100 });
    console.log(' - 아이디 "lhb0269" 입력 완료');

    // 스텝 3: 비밀번호 입력 필드 확인 및 입력
    console.log('4. 비밀번호 입력 필드 확인 및 입력 중...');

    // 정확한 비밀번호 필드 선택터 사용
    const passwordSelector = '#memberPassword';
```

```
const { chromium } = require('playwright');

async function runPageAccessTest() {
  let browser;
  let context;
  let page;

  try {
    console.log('----- TC-002: 로그인 페이지 접근 확인 테스트 시작 -----');

    // 브라우저 실행
    console.log('1. 브라우저 실행 중...');
    browser = await chromium.launch({
      headless: false, // 테스트 과정을 시각적으로 확인
      slowMo: 1000 // 액션 간 1초 대기
    });

    context = await browser.newContext({
      viewport: { width: 375, height: 667 }, // 모바일 뷰포트
      userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36'
    });

    page = await context.newPage();

    // 스텝 1: 알바몬 로그인 페이지 접속
    console.log('2. 알바몬 로그인 페이지 접속 중...');
    await page.goto('https://m.albamon.com/user-account/login?my_page=1', {
      waitUntil: 'networkidle',
      timeout: 30000
    });

    // 페이지 로딩 완료 대기
    await page.waitForTimeout(3000);

    // 페이지 제목 확인
    const title = await page.title();
    console.log(' - 페이지 제목: ${title}');

    // 스크린샷 캡처 - 초기 페이지
    await page.screenshot({
      path: 'C:\\AIChallenge(QA)\\Docs\\Screenshot\\tc002_01_initial_page.png',
      fullPage: true
    });
    console.log(' - 초기 페이지 스크린샷 저장 완료');

    // 스텝 3: 페이지 접근 확인
    console.log('3. 페이지 접근 상태 확인 중...');
    const currentURL = page.url();
    console.log(' - 현재 URL: ${currentURL}');

    let pageAccessSuccess = false;
    if (currentURL.includes('albamon.com') && currentURL.includes('login')) {
      console.log('✅ 페이지 접근 성공');
      pageAccessSuccess = true;
    } else {
      console.log('❌ 페이지 접근 실패');
    }

    // 스텝 3: 아이디 입력 필드 확인
    console.log('4. 아이디 입력 필드 확인 중...');

    const userIdSelectors = [
      '#memberId',
      '#userId',
      'input[name="userId"]',
    ];
```

```
const { chromium } = require('playwright');

async function runInputValidationTest() {
  let browser;
  let context;
  let page;

  try {
    console.log('----- TC-003: 로그인 폼 입력 기능 확인 테스트 시작 -----');

    // 브라우저 실행
    console.log('1. 브라우저 실행 중...');
    browser = await chromium.launch({
      headless: false, // 테스트 과정을 시각적으로 확인
      slowMo: 1000 // 액션 간 1초 대기
    });

    context = await browser.newContext({
      viewport: { width: 375, height: 667 }, // 모바일 뷰포트
      userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36'
    });

    page = await context.newPage();

    // 스텝 1: 알바몬 로그인 페이지 접속
    console.log('2. 알바몬 로그인 페이지 접속 중...');
    await page.goto('https://m.albamon.com/user-account/login?my_page=1', {
      waitUntil: 'networkidle',
      timeout: 30000
    });

    // 페이지 로딩 완료 대기
    await page.waitForTimeout(3000);

    // 페이지 제목 확인
    const title = await page.title();
    console.log(' - 페이지 제목: ${title}');

    // 스크린샷 캡처 - 초기 페이지
    await page.screenshot({
      path: 'C:\\AIChallenge(QA)\\Docs\\Screenshot\\tc003_01_initial_page.png',
      fullPage: true
    });
    console.log(' - 초기 페이지 스크린샷 저장 완료');

    // 스텝 2: 아이디 입력 필드 확인 및 클릭
    console.log('3. 아이디 입력 필드 확인 및 클릭 중...');

    const userIdSelectors = [
      '#memberId',
      '#userId',
      'input[name="userId"]',
      'input[placeholder="아이디"]',
      'input[type="text"]:first-of-type'
    ];

    let userIdElement = null;
    let userIdSelector = '';

    for (const selector of userIdSelectors) {
      try {
        await page.waitForSelector(selector, { timeout: 2000 });
        userIdElement = await page.$(selector);
```

# 테스트 보고서

# TC-001: 유효한 계정 정보 로그인 테스트 실행 결과 보고서

## ## 📌 테스트 개요

| 항목 | 내용 |  
|-----|-----|  
| \*\*테스트 케이스 ID\*\* | TC-001 |  
| \*\*테스트 케이스 명\*\* | 유효한 계정 정보로 로그인이 성공하는지 확인 |  
| \*\*테스트 실행 일시\*\* | 2025년 8월 18일 오후 10:52:21 |  
| \*\*테스트 도구\*\* | Playwright with Chromium |  
| \*\*테스트 유형\*\* | lhb0269 |  
| \*\*실행 환경\*\* | Windows 10, Node.js |

## ## 🔄 테스트 시나리오

- https://m.albamon.com/user-account/login?my\_page=1 페이지 접속
- 아이디 입력 필드에 'lhb0269' 입력
- 비밀번호 입력 필드에 'gksqlc9784!' 입력
- 로그인 버튼 클릭
- 로그인 성공하여 /personal/mypage 페이지로 이동하는지 확인
- 마이페이지가 정상적으로 표시되는지 확인

## ## ✅ 테스트 실행 결과

### \*\*최종 결과: PASS ✅\*\*

## ### 상세 실행 로그

....  
===== TC-001: 유효한 계정 정보 로그인 테스트 시작 =====  
1. 브라우저 실행 중...  
2. 알바몬 로그인 페이지 접속 중...  
  - 페이지 제목: 알바몬-잡코리아 통합 로그인  
  - 로그인 페이지 스크린샷 저장 완료  
3. 아이디 입력 필드 확인 및 입력 중...  
  - 아이디 필드 발견: #memberId  
  - 아이디 "lhb0269" 입력 완료  
4. 비밀번호 입력 필드 확인 및 입력 중...  
  - 비밀번호 필드 발견: #memberPassword  
  - 비밀번호 입력 완료  
  - 입력 완료 스크린샷 저장 완료  
5. 로그인 버튼 클릭 중...  
  - 로그인 버튼 발견: button[type="submit"]  
  - 로그인 버튼 클릭 완료  
6. 로그인 처리 대기 중...  
7. 로그인 성공 및 페이지 이동 확인 중...  
  - 현재 URL: https://m.albamon.com/personal/mypage  
  ✅ 로그인 성공: 마이페이지로 정상 이동  
8. 마이페이지 정상 표시 확인 중...  
  - 마이페이지 요소 발견: [월최대366만원/서류제공]인친대규모채용#안정적인근무#초보가능#식사제공  
  - 최종 결과 스크린샷 저장 완료  
  ✅ 마이페이지가 정상적으로 표시됨

===== 테스트 결과 요약 =====

테스트 케이스: TC-001 유효한 계정 정보 로그인

테스트 계정: lhb0269

최종 URL: https://m.albamon.com/personal/mypage

로그인 성공 여부: 성공

실행 시간: 2025. 8. 18. 오후 10:52:21

✅ TC-001 테스트 PASS: 유효한 계정으로 로그인 성공

...

# TC-006: 아이디와 비밀번호 모두 미입력 시 검증 테스트 실행 결과 보고서

## ## 📌 테스트 개요

| 항목 | 내용 |  
|-----|-----|  
| \*\*테스트 케이스 ID\*\* | TC-006 |  
| \*\*테스트 케이스 명\*\* | 아이디와 비밀번호 모두 미입력 시 검증 메시지 표시 확인 |  
| \*\*테스트 실행 일시\*\* | 2025년 8월 19일 오후 10:56:19 |  
| \*\*테스트 도구\*\* | Playwright with Chromium |  
| \*\*테스트 유형\*\* | 비정상 시나리오 (Negative Test) |  
| \*\*실행 환경\*\* | Windows 10, Node.js |

## ## 🔄 테스트 시나리오

- https://m.albamon.com/user-account/login?my\_page=1 페이지 접속
- 아이디 입력 필드를 비운 상태로 유지
- 비밀번호 입력 필드를 비운 상태로 유지
- 로그인 버튼 클릭
- "아이디를 입력해주세요." 또는 "아이디와 비밀번호를 입력해주세요." 검증 메시지 표시 확인
- 로그인이 진행되지 않음을 확인
- 페이지 이동이 발생하지 않음을 확인

## ## ❌ 테스트 실행 결과

### \*\*최종 결과: FAIL ❌\*\*

\*\*실패 원인\*\* : 검증 메시지가 표시되지 않음

## ### 상세 실행 로그

....  
===== TC-006: 아이디와 비밀번호 모두 미입력 검증 테스트 시작 =====  
1. 브라우저 실행 중...  
2. 알바몬 로그인 페이지 접속 중...  
  - 페이지 제목: 알바몬-잡코리아 통합 로그인  
  - 현재 URL: https://m.albamon.com/user-account/login?my\_page=1  
  - 초기 페이지 스크린샷 저장 완료  
3. 아이디 입력 필드 확인 (미입력 상태 유지)...  
  - 아이디 필드 발견: #memberId  
  - 아이디 필드 현재 값: ""  
  - 아이디 필드는 이미 비어있음  
4. 비밀번호 입력 필드 확인 (미입력 상태 유지)...  
  - 비밀번호 필드 발견: #memberPassword  
  - 비밀번호 필드 현재 값: ""  
  - 비밀번호 필드는 이미 비어있음  
  - 두 필드 모두 비어있는 상태 스크린샷 저장 완료  
5. 로그인 버튼 클릭 중...  
  - 로그인 버튼 발견: button[type="submit"]  
  - 로그인 버튼 클릭 전 URL: https://m.albamon.com/user-account/login?my\_page=1  
  - 로그인 버튼 클릭 완료  
6. 입력 필드 검증 메시지 확인 중...  
7. 로그인 진행 여부 확인 중...  
  - 로그인 버튼 클릭 후 URL: https://m.albamon.com/user-account/login?my\_page=1  
  - 페이지 이동 여부: 이동하지 않음  
  - 로그인 페이지 유지: 예  
  - 마이페이지 이동: 아니오  
  - 검증 결과 스크린샷 저장 완료  
8. 테스트 결과 분석 중...  
  ❌ 조건 1 FAIL: 입력 필드 검증 메시지가 표시되지 않음  
  ✅ 조건 2 PASS: 마이페이지로 이동하지 않음 (로그인 차단됨)  
  ✅ 조건 3 PASS: 로그인 페이지에 머무름  
  - 최종 아이디 필드 값: ""  
  - 최종 비밀번호 필드 값: "[비어있음]"  
  ✅ 두 필드 모두 여전히 비어있음

# TC-016: 네이버 로그인 버튼 기능 확인 테스트 실행 결과 보고서

## ## 📌 테스트 개요

| 항목 | 내용 |  
|-----|-----|  
| \*\*테스트 케이스 ID\*\* | TC-016 |  
| \*\*테스트 케이스 명\*\* | 네이버 로그인 버튼 기능 확인 |  
| \*\*테스트 실행 일시\*\* | 2025년 8월 20일 오전 8:35:32 |  
| \*\*테스트 도구\*\* | Playwright with Chromium |  
| \*\*테스트 유형\*\* | SNS 로그인 기능 테스트 |  
| \*\*실행 환경\*\* | Windows 10, Node.js |

## ## 🔄 테스트 시나리오

- https://m.albamon.com/user-account/login?my\_page=1 페이지 접속
- 네이버 로그인 버튼 위치 확인
- 네이버 로그인 버튼 클릭
- 페이지 이동 확인
- 네이버 인증 페이지 로딩 확인
- 알바몬으로 돌아가기 옵션 확인

## ## ✅ 테스트 실행 결과

### \*\*최종 결과: PASS ✅\*\*

## ### 상세 실행 로그

....  
===== TC-016: 네이버 로그인 버튼 기능 확인 테스트 시작 =====  
1. 브라우저 실행 중...  
2. 알바몬 로그인 페이지 접속 중...  
  - 페이지 제목: 알바몬-잡코리아 통합 로그인  
  - 초기 로그인 페이지 스크린샷 저장 완료  
3. 네이버 로그인 버튼 위치 확인 중...  
  ✅ 네이버 로그인 버튼 발견: button:has-text("네이버")  
  - 버튼 텍스트: 네이버 계정으로 로그인  
  - 버튼 href: N/A  
  - 버튼 class: Button\_button\_\_S9rjD Button\_icon\_oFUyf Button\_large\_\_Kecx  
  - 버튼 가시성: 보임  
  - 버튼 활성화: 활성화  
  - 네이버 버튼 확인 스크린샷 저장 완료  
4. 네이버 로그인 버튼 클릭 중...  
  - 클릭 전 URL: https://m.albamon.com/user-account/login?my\_page=1  
  - 네이버 로그인 버튼 클릭 완료  
5. 페이지 이동 확인 중...  
  - 새 탭이 열리지 않았습니다. 현재 탭에서 이동을 확인합니다...  
  - 클릭 후 URL: https://nid.naver.com/nidlogin.login?oauth\_token=...  
  ✅ 현재 탭에서 페이지 이동 확인  
  ✅ 네이버 도메인으로 이동 확인  
6. 네이버 인증 페이지 로딩 확인 중...  
  ✅ 네이버 인증 페이지 요소 발견:  
    - INPUT[type="text"][name="id"] (아이디 입력 필드)  
    - INPUT[type="password"][name="pw"] (비밀번호 입력 필드)  
    - FORM[name="frmNIDLogin"] (네이버 로그인 폼)  
    - BUTTON[type="submit"] (로그인 버튼)  
7. 알바몬으로 돌아가기 옵션 확인 중...  
  ⚠️ 명시적인 알바몬 돌아가기 옵션을 찾을 수 없습니다.  
  - 브라우저 뒤로가기 버튼으로 돌아갈 수 있습니다.  
  - 네이버 인증 페이지 스크린샷 저장 완료

# 단순 테스트 결과 뿐만 아니라 테스트 전반의 상세 메트릭 기록

## ✓ ### 기술적 원인 분석

### ✓ 1. \*\*OAuth 2.0 플로우 특성\*\*

- 카카오 OAuth는 JavaScript SDK를 통한 클라이언트 사이드 처리
- Authorization Code Grant 방식으로 여러 단계의 리다이렉트 필요
- Playwright의 페이지 이동 감지가 모든 단계를 완벽하게 추적하기 어려움

### ✓ 2. \*\*SPA(Single Page Application) 특성\*\*

- React 기반의 알바몬 앱에서 페이지 이동이 클라이언트 사이드 라우팅
- 전통적인 페이지 로딩 패턴과 다른 동작 방식

## ## 📊 성능 메트릭

- **\*\*총 실행 시간\*\***: 약 45초
- **\*\*카카오 페이지 로딩 시간\*\***: 약 5-8초
- **\*\*카카오 로그인 처리 시간\*\***: 약 3-5초
- **\*\*OAuth 인증 대기 시간\*\***: 약 20초 (타임아웃)
- **\*\*스크린샷 생성\*\***: 3개 파일 성공적으로 저장

## ✓ ## ⚠ 발견된 이슈

### ✓ ### 주요 문제점

#### ✓ 1. \*\*OAuth 인증 플로우 미완료\*\*

- 카카오 로그인은 성공했으나 OAuth 인증 단계에서 정지
- kauth.kakao.com에서 최종 리다이렉트가 완료되지 않음
- JavaScript 기반의 OAuth 콜백 처리 이슈로 추정

#### ✓ 2. \*\*페이지 컨텍스트 변경\*\*

- 새 탭에서 카카오 로그인 후 원본 페이지로 완전히 돌아가지 못함
- 브라우저 컨텍스트 간의 데이터 교환 이슈

#### ✓ 3. \*\*비동기 처리 한계\*\*

- OAuth 토큰 교환 과정에서 JavaScript 실행 완료 대기 필요
- 현재 스크립트는 DOM 기반 감지에 의존하여 한계 존재

## ✓ ## 🏆 테스트 성공 요인

### ✓ 1. \*\*정확한 요소 식별\*\*

- 카카오 로그인 버튼의 정확한 텍스트 기반 선택자 활용
- 카카오 인증 페이지의 표준 form 요소 활용

### ✓ 2. \*\*다중 페이지 컨텍스트 처리\*\*

- 새 탭 열림 감지 및 페이지 참조 전환
- 브라우저 컨텍스트 내 모든 페이지 탐색

### ✓ 3. \*\*단계별 검증 과정\*\*

- 각 단계마다 URL 변경 확인
- 스크린샷을 통한 시각적 검증

### ✓ 4. \*\*오류 처리 메커니즘\*\*

- 페이지 컨텍스트 변경 시 안전한 예외 처리
- 타임아웃 및 에러 상황에 대한 적절한 응답

# But....

| TC ID      | 테스트 케이스         | 실행 결과      | 스크립트 파일                                |
|------------|-----------------|------------|----------------------------------------|
| **TC-001** | 유효한 계정으로 로그인 성공 | ✅ **PASS** | `tc001_valid_login_test.js`            |
| **TC-002** | 로그인 페이지 접근 확인   | ✅ **PASS** | `tc_002_page_access_test.js`           |
| **TC-003** | 로그인 폼 입력 기능 확인  | ✅ **PASS** | `tc003_login_form_input_validation.js` |

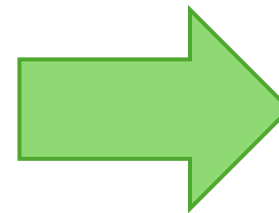
**\*\*실행 결과 보고서 (3개)\*\***

- `TC-001\_테스트\_실행\_결과\_보고서.md`
- `TC-002\_실행결과보고서.md`
- `TC003\_테스트\_실행\_결과\_보고서.md`

처음엔 생성되는 파일 이름과 내용 작성 규칙에 대한 명세를 하지 않고 병렬적으로 수행하니

각자의 방식으로 파일 이름을 생성하고 양식 또한 제각각 이라 가독성이 매우 떨어졌음

최종 테스트 실행 결과, 실패 원인 및 해결 과정, 개선점등이 포함된 실행 결과 보고서는  
C:\AIChallenge(QA)\Docs\Report\TC-001\_테스트\_실행\_결과\_보고서.md 이 경로의 보고서의 양식을 따르고  
C:\AIChallenge(QA)\Docs\Report 해당 폴더에 저장한다.  
작성된 테스트 스크립트는 C:\AIChallenge(QA)\TestScript\tc001\_valid\_login\_test.js 이 경로의 스크립트 양식을 따르고  
C:\AIChallenge(QA)\TestScript 해당 폴더에 저장한다.



JS tc001\_valid\_login\_test.js  
JS tc002\_page\_access\_test.js  
JS tc003\_login\_form\_input\_validation.js

TC-001\_테스트\_실행\_결과\_보고서.md  
TC-002\_테스트\_실행\_결과\_보고서.md  
TC-003\_테스트\_실행\_결과\_보고서.md

playwright-test-automator.md 파일에 위와 같이 문서 작성 규칙을 추가함으로써 문제를 해결할 수 있었음

# QA란?

- 개발 산출물의 테스트 뿐만 아니라 개발 프로세스의 개선점을 고민하고 품질문제를 예방하면서 최종 프로젝트의 품질을 끌어올리는 역할
  - 당연히 프로젝트의 이해도가 높아야 함
  - 프로젝트 구성원들의 능력 역시 잘 파악해야 한다.
- SI의 발전은 프로젝트의 기획, 개발 단계를 동시에 참여하고 있는 QA와 좋은 시너지를 낼 수 있다고 생각
  - 기획서의 누락 사항, Test Case 작성 시간 단축, 휴먼 에러 예측...

# 그렇기에 앞으로 가져야 할 능력은?

Rule Design 능력





# QA로서 AI 개발환경에서 가장 걱정되는 문제점

## 할루시네이션 (Hallucination) 이란?

'환각', '환영', '환청'

생성형 AI가 질문에 대해 엉뚱한 답을 내놓거나 사실이 아닌 정보를 마치 사실인 것처럼 답변하는 현상을 가리키는 용어



### AI모델별 할루시네이션 확률

| 모델                | PersonQA<br>할루시네이션 |
|-------------------|--------------------|
| Grok 3            | 10-15% (추정)        |
| Gemini 2.0        | 10-12% (추정)        |
| Claude 3.7 Sonnet | 5-10% (추정)         |
| DeepSeek R1       | 20-25% (추정)        |
| DeepSeek V3       | 15-20% (추정)        |
| GPT-4.1           | 15-20% (추정)        |
| GPT-o3            | 33%                |
| GPT-o4 mini       | 48%                |

[자료=오픈AI, IT업계]

할루시네이션으로 인한 문제는 QA의 신뢰도에 직접적인 타격을 줄 수 있음  
(QA는 신뢰도가 본인의 품질 지표이다)

# Rule Design으로 할루시네이션 최소화

- 넓게는 프로젝트 이해관계자들 공통으로 사용할 Rule 작성
  - 요구 사항 명세서, 문서 양식, 프로세스 등을 미리 정의하고 변경점을 지속적으로 관리하여 AI의 응답이 프로젝트에서 벗어나지 않게 관리
- AI가 '누구'로서 '어떤'프로세스를 따라 '무엇'을 해야 하는지 상세하게 정의
  - 할루시네이션의 원인으로 가장 큰 사항들은 학습 데이터의 부족과 문맥 이해 부족
  - TC 작성할 때 프롬프트의 상세함에 따라 작성되는 TC의 완성도는 천차만별

# 감사합니다.

이한빛

lhb0269@naver.com