Cypress Tesztautomatizálás - Tippek és Trükkök

Alapfogalmak és elméleti háttér

Mi a Mocha?

A Mocha egy JavaScript tesztkeretrendszer, amely a Cypress alapjait képezi. A Mocha felelős a tesztstruktúra kialakításáért - vagyis a `describe()` és `it()` blokkok működéséért, a tesztek csoportosításáért és futtatásáért. A Cypress gyakorlatilag átveszi a Mocha nyelvezetét és kibővíti saját böngészőautomatizálási képességekkel.

A Mocha biztosítja az aszinkron tesztek támogatását, a hook rendszert (before, beforeEach, after, afterEach) és a tesztek futtatásának rugalmasságát. Fontos megérteni, hogy amikor Cypressben describe/it blokkokat használunk, valójában a Mocha tesztstruktúrájában dolgozunk, amit a Cypress saját parancsaival egészít ki.

Mi a Cypress Hook?

A hookok olyan speciális függvények, amelyek a tesztek életciklusának különböző pontjain futnak le automatikusan. Ezek lehetővé teszik, hogy előkészítő vagy takarító műveleteket végezzünk a tesztek előtt vagy után.

A hookok típusai:

- **before()**: Egyszer fut le az összes teszt előtt egy describe blokkban
- **beforeEach()**: Minden egyes teszt előtt lefut
- **after()**: Egyszer fut le az összes teszt után
- **afterEach()**: Minden egyes teszt után lefut

A globális hookok a support fájlokban definiálhatók és minden tesztre érvényesek lesznek. A plugin hookok pedig a cypress.config.js fájlban definiálhatók és a Cypress életciklusának eseményeire reagálnak (például böngésző indítása, screenshot készítése).

Queue és Retry mechanizmus

A Cypress parancs queue (sor) egy kulcsfontosságú koncepció. Amikor Cypress parancsokat írunk, azok nem azonnal futnak le, hanem egy belső sorba kerülnek. Ez a sor aztán szekvenciálisan dolgozza fel a parancsokat, közben kezelve az automatikus várakozást és újrapróbálkozást.

A retry mechanizmus azt jelenti, hogy minden Cypress parancs automatikusan újrapróbálkozik egy meghatározott ideig (alapértelmezetten 4 másodperc), amíg a feltétel teljesül. Ez kiküszöböli a

manuális várakozások szükségességét - nem kell sleep vagy wait parancsokat írnunk, mert Cypress okosan vár amíg az elem megjelenik vagy az állapot elérhető lesz.

Az Alias szerepe

Az alias egy becenév vagy címke, amit Cypress elemeknek, értékeknek vagy hálózati hívásoknak adhatunk. Az alias-ok lehetővé teszik a kód újrafelhasználását és tisztábbá tételét.

Az alias-ok különösen hasznosak hálózati intercept-eknél, ahol megnevezhetjük az API hívásokat és később várakozhatunk rájuk. Ez stabilitást ad a teszteknek, mert pontosan tudjuk, mikor fejeződött be egy hálózati művelet. Az alias-ok emellett elemek tárolására is használhatók, így nem kell többször ugyanazt a szelektort megírni.

Cypress Custom Commands

A custom commandok olyan saját parancsok, amelyeket a Cypress API-hoz adunk hozzá. Ezek a parancsok ugyanúgy láncolhatók és használhatók, mint a beépített Cypress parancsok.

Miért jó a Custom Command:

- Kód újrafelhasználhatóság: egyszer definiáljuk, sokszor használjuk
- Absztrakció: komplex műveletek egyszerű parancsokba rejtése
- Karbantarthatóság: egy helyen kell módosítani, ha változik a logika
- Olvashatóság: a tesztek természetes nyelvhez hasonlóan olvashatók
- Csapat szabványok: egységes megközelítés a gyakori műveletekhez

Hogyan működik: A custom commandok a `Cypress.Commands.add()` metódussal definiálhatók a support fájlokban. Ezek a parancsok bekerülnek a Cypress láncolható API-jába és ugyanúgy viselkednek, mint a natív parancsok - támogatják a retry mechanizmust, alias-okat és a queue rendszert.

Mikor ne használjunk Custom Commandokat:

Több helyzet van, amikor kerülni kell a custom commandok használatát:

- **Egyedi, csak egyszer használt műveleteknél**: Ha egy parancs csak egyetlen tesztben fordul elő, felesleges globálisan definiálni
- **Túlzott absztrakció esetén**: Ha a custom command elrejti a fontos tesztlogikát és nehézzé teszi a megértést
- **Túl általános parancsok**: Amikor egy command túl sokféle dolgot csinál és nehéz megérteni a célját

- **Assertion-ök elrejtése**: A tesztellenőrzéseket inkább a tesztben kell tartani a jobb átláthatóság érdekében

Best Practice-k Custom Commandokhoz:

- **Egy felelősség egy parancs**: Minden custom command egyetlen, jól definiált feladatot végezzen
- **Beszédes elnevezés**: A parancs neve egyértelműen fejezze ki, mit csinál
- **Dokumentáció**: Minden custom commandhoz írjunk rövid leírást
- **Stabil szelektorok**: Ne CSS osztályokra építsünk, hanem data-testid attribútumokra
- **Paraméterezhetőség**: A commandok legyenek rugalmasak paraméterekkel
- **Ne rejtsünk el mindent**: A fő logikai ellenőrzések maradjanak a tesztekben

Jó tesztadatok jellemzői

A jó tesztadat meghatározza a tesztek megbízhatóságát és karbantarthatóságát:

Determinisztikus: Mindig ugyanazt az eredményt produkálja, függetlenül a futtatás idejétől vagy környezetétől. Kerüljük a véletlenszerű értékeket, hacsak nem specifikusan azt teszteljük.

Reális: Tükrözzék a valós felhasználási eseteket. Ne csak "test", "123" típusú dummy adatokat használjunk, hanem olyan értékeket, amelyek valóban előfordulhatnak éles környezetben.

Izolált: Minden teszt saját adatkészlettel dolgozzon, ne függjön más tesztek eredményeitől. Ez biztosítja, hogy a tesztek tetszőleges sorrendben futtathatók.

Konzisztens: Ugyanazt az adatstruktúrát és formátumot használjuk a hasonló tesztesetekben. Ez megkönnyíti a karbantartást és a hibakeresést.

Értelmezhető: Az adatok jelentése legyen egyértelmű a teszt kontextusában. Használjunk beszédes változóneveket és értékeket.

Hasznos Cypress Plugins

Image-Snapshot Plugin

Ez a plugin vizuális regressziós tesztelést tesz lehetővé. Képernyőképeket készít az alkalmazásról és összehasonlítja őket referencia képekkel. Ha vizuális eltérést észlel, a teszt megbukik.

A plugin különösen hasznos olyan alkalmazásoknál, ahol fontos a vizuális megjelenés konzisztenciája. Automatikusan észleli a nem szándékolt design változásokat, betűtípus módosításokat vagy layout problémákat. A plugin rugalmasan konfigurálható - beállíthatjuk a tolerancia szintet és kizárhatunk változó elemeket (például időbélyegeket).

Mochawesome Reporter

Ez egy haladó HTML jelentés generátor, amely szép, részletes és interaktív riportokat készít a tesztfutásokról. A riportok tartalmazzák a tesztek státuszát, futási időket, screenshot-okat és hibaüzeneteket.

A Mochawesome különösen értékes CI/CD környezetben, ahol a teszteredmények megosztása és elemzése fontos. A riportok diagramokat is tartalmaznak a tesztek megoszlásáról és trend információkat nyújtanak. A plugin beágyazhatja a screenshot-okat és videókat közvetlenül a riportba.

Cypress-XPath Plugin

Alapértelmezetten a Cypress csak CSS szelektorokat támogat, ez a plugin XPath szelektorok használatát teszi lehetővé. Az XPath hasznos olyan esetekben, ahol a CSS szelektorok nem elegendőek - például szöveg alapú elemkereséshez vagy komplex DOM struktúrák navigálásához.

Bár a CSS szelektorok általában gyorsabbak és modernebbek, bizonyos legacy rendszereknél vagy speciális igényeknél az XPath rugalmasabb megoldást nyújt. A plugin zökkenőmentesen integrálja az XPath támogatást a Cypress parancs láncolásába.

Accessibility (Akadálymentesség) Pluginok

Az akadálymentességi pluginok automatikusan ellenőrzik, hogy az alkalmazás megfelel-e az akadálymentességi szabványoknak (WCAG). Ezek a pluginok olyan problémákat észlelnek, mint a hiányzó alt attribútumok, rossz kontrasztarányok vagy nem megfelelő ARIA labelek.

Az akadálymentesség vizsgálata különösen fontos olyan alkalmazásoknál, amelyeket széles felhasználói kör használ. A pluginok segítenek automatizálni ezeket az ellenőrzéseket és korai szakaszban felhívják a figyelmet a problémákra, amikor még könnyebb javítani őket.

Mi a Cypress és miért speciális?

A Cypress egy modern end-to-end tesztelő keretrendszer, amely számos egyedülálló jellemzővel rendelkezik:

Cypress sajátosságok

- **Böngészőben fut**: A tesztek közvetlenül a böngészőben futnak, nem kívülről vezérlik
- **Automatikus várakozás**: Nem kell explicit wait-eket írni - Cypress automatikusan vár az elemekre
- **Időutazás**: Visszaléphet a tesztvégrehajtás során és megtekintheti, mi történt
- **Valós idejű újratöltés**: A tesztek automatikusan újrafutnak mentéskor
- **Screenshot és videó**: Automatikus képernyőképek hibák esetén

- **Parancs láncolás**: cy.get().should().click() stílusú szintaxis

Miben jó a Cypress?

- **E2E tesztelés**: Teljes felhasználói útvonalak tesztelése
- **API tesztelés**: cy.request() paranccsal
- **Komponens tesztelés**: React, Vue, Angular komponensek izolált tesztelése
- **Vizuális regresszió**: Pluginokkal képernyőkép-összehasonlítás
- **Fejlesztői élmény**: Kiváló debugging és hibajelzés

Cypress buktatói és korlátai

Főbb limitációk

- **Csak Chromium alapú böngészők**: Firefox támogatás korlátozott
- **Egy domain**: Alapértelmezetten egy domainre korlátozódik (cy.origin() kivétel)
- **Multiple tabs**: Nem tud több fül között váltani
- **Mobile testing**: Natív mobilalkalmazásokat nem tud tesztelni
- **Performance**: Nagyobb projekteknél lassabb lehet mint más eszközök

Gyakori hibák

- **Sync vs Async confusion**: Cypress parancsok aszinkronok, de a szintaxis szinkronnak tűnik
- **Element visibility**: CSS-sel elrejtett elemek problémái
- **Flaky tesztek**: Instabil szelektorok és időzítési problémák

Flakiness elkerülése

1. Stabil szelektorok használata

javascript

```javascript
// Rossz - törékeny szelektorok
cy.get('.btn-primary:nth-child(3)')
cy.get('#element-1234567890')

// Jó - stabil szelektorok
cy.get('[data-testid="submit-button"]')
cy.get('[data-cy="user-menu"]')
```

2. Automatikus várakozás kihasználása

```javascript
// Rossz - fix várakozás
cy.wait(2000)
cy.get('.loading').should('not.exist')

// Jó - feltétel alapú várakozás
cy.get('.loading').should('not.exist')
cy.get('[data-testid="results"]').should('be.visible')
```

3. Alias használata hálózati hívásokhoz

```javascript
cy.intercept('GET', '/api/users').as('getUsers')
cy.visit('/users')
cy.wait('@getUsers')
cy.get('[data-testid="user-list"]').should('exist')
```

4. Környezeti változók kezelése

```javascript
// cypress.config.js
baseUrl: process.env.CYPRESS_BASE_URL || 'http://localhost:3000'

// Tesztben
const apiUrl = Cypress.env('API_URL')
```

User Journey implementálása

1. Moduláris megközelítés

```javascript
```

```javascript
// Felosztás kisebb, újrafelhasználható lépésekre
Cypress.Commands.add('loginUser', (email, password) => {
  cy.visit('/login')
  cy.get('[data-testid="email"]').type(email)
  cy.get('[data-testid="password"]').type(password)
  cy.get('[data-testid="submit"]').click()
  cy.url().should('include', '/dashboard')
})

// Használat
describe('User Journey', () => {
  it('teljes vásárlási folyamat', () => {
    cy.loginUser('test@example.com', 'password')
    cy.addProductToCart('laptop')
    cy.proceedToCheckout()
    cy.fillShippingDetails()
    cy.completePayment()
  })
})
```

2. Session kezelés optimalizálása

javascript

```javascript
// Gyors bejelentkezés session cache-sel
beforeEach(() => {
  cy.session('user-session', () => {
    cy.loginUser('test@example.com', 'password')
  })
})
```

3. Test Data Management

javascript

```
// Fixture használata
cy.fixture('users.json').then((users) => {
  cy.loginUser(users.testUser.email, users.testUser.password)
})

// Dinamikus adatok
beforeEach(() => {
  cy.task('createTestUser').then((user) => {
    cy.wrap(user).as('testUser')
  })
})
```

Best Practices

1. Test Organization

```
cypress/
├── e2e/
│   ├── auth/
│   │   ├── login.cy.js
│   │   └── registration.cy.js
│   ├── shopping/
│   │   ├── product-search.cy.js
│   │   └── checkout.cy.js
├── fixtures/
│   ├── users.json
│   └── products.json
├── support/
│   ├── commands/
│   │   ├── auth.js
│   │   └── ui.js
│   ├── commands.js
│   └── e2e.js
```

2. Page Object Pattern alternatíva

javascript

```
// App actions pattern (ajánlott Cypress-nél)
const loginActions = {
  visit: () => cy.visit('/login'),
  fillForm: (email, password) => {
    cy.get('[data-testid="email"]').type(email)
    cy.get('[data-testid="password"]').type(password)
  },
  submit: () => cy.get('[data-testid="submit"]').click()
}

// Használat
it('bejelentkezés', () => {
  loginActions.visit()
  loginActions.fillForm('user@test.com', 'password')
  loginActions.submit()
})
```

3. Error Handling

javascript

```
// Global hibakezelés
Cypress.on('uncaught:exception', (err) => {
  // Ignorál bizonyos hibákat
  if (err.message.includes('ResizeObserver')) {
    return false
  }
})

// Teszten belül
cy.get('[data-testid="submit"]')
  .click()
  .then(() => {
    // Siker ág
  })
  .catch(() => {
    // Hiba ág - alternatív útvonal
  })
```

API Tesztelés Cypress-szel

1. Alap API hívások

javascript

```javascript
it('API endpoint tesztelése', () => {
  cy.request({
    method: 'POST',
    url: '/api/users',
    body: {
      name: 'Test User',
      email: 'test@example.com'
    }
  }).then((response) => {
    expect(response.status).to.eq(201)
    expect(response.body).to.have.property('id')
  })
})
```

2. Paraméterezett tesztelés

javascript

```javascript
const testCases = [
  { input: 'valid@email.com', expected: 200 },
  { input: 'invalid-email', expected: 400 },
  { input: '', expected: 400 }
]

testCases.forEach(({ input, expected }) => {
  it(`email validáció: ${input}`, () => {
    cy.request({
      method: 'POST',
      url: '/api/validate-email',
      body: { email: input },
      failOnStatusCode: false
    }).its('status').should('eq', expected)
  })
})
```

Custom Commands és Plugins

1. Saját parancsok létrehozása

javascript

```javascript
// cypress/support/commands.js
Cypress.Commands.add('getByTestId', (testId) => {
  return cy.get(`[data-testid="${testId}"]`)
})

Cypress.Commands.add('loginViaAPI', (email, password) => {
  return cy.request('POST', '/api/login', { email, password })
    .then((response) => {
      cy.setCookie('auth-token', response.body.token)
    })
})
```

2. Hasznos pluginok

- **cypress-xpath**: XPath szelektorok támogatása
- **cypress-mochawesome-reporter**: Szép HTML riportok
- **cypress-image-snapshot**: Vizuális regresszió tesztelés
- **@badeball/cypress-cucumber-preprocessor**: BDD/Gherkin támogatás

CI/CD Integráció

1. GitHub Actions példa

yaml

```yaml
name: Cypress Tests
on: [push, pull_request]

jobs:
  cypress:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: cypress-io/github-action@v6
        with:
          browser: chrome
          start: npm start
          wait-on: 'http://localhost:3000'
        env:
          CYPRESS_RECORD_KEY: ${{ secrets.CYPRESS_RECORD_KEY }}
```

2. Secretek kezelése

javascript

```javascript
// cypress.config.js
env: {
  API_URL: process.env.CYPRESS_API_URL,
  API_TOKEN: process.env.CYPRESS_API_TOKEN
}

// Tesztben
const token = Cypress.env('API_TOKEN')
```

Debugging és Troubleshooting

1. Debug technikák

javascript

```
// Konzol logolás
cy.get('[data-testid="element"]').then(($el) => {
  console.log($el)
  cy.log('Element text:', $el.text())
})

// Pause és inspect
cy.get('[data-testid="form"]').pause()
cy.debug()

// Screenshot készítése
cy.screenshot('debug-screenshot')
```

2. Cypress Studio használata

```
javascript

// cypress.config.js
experimentalStudio: true
```

3. Gyakori problémák megoldása

- **Element not found**: Várj az elem megjelenésére
- **Element not interactable**: Ellenőrizd, hogy látható és kattintható-e
- **CORS errors**: baseUrl és chromeWebSecurity beállítások
- **Timeout errors**: Növeld a defaultCommandTimeout értéket

Performance és Optimalizáció

1. Teszt futási idő csökkentése

```
javascript

// Session cache használata
cy.session('user', loginFunction, { cacheAcrossSpecs: true })

// Párhuzamos futtatás
// cypress.config.js
video: false, // gyorsabb futás
screenshotOnRunFailure: false
```

2. Memory management

```javascript
// Cleanup after tests
afterEach(() => {
  cy.clearCookies()
  cy.clearLocalStorage()
})
```

Összefoglalás

A Cypress egy hatékony eszköz, de fontos megérteni a korlátait és best practice-eket. A kulcs a stabil szelektorok használatában, a proper várakozási stratégiákban és a jó tesztszervezésben rejlik. Az automatikus várakozás és retry mechanizmus kihasználásával jelentősen csökkenthető a flaky tesztek száma.

Legfontosabb tanácsok:

1. Használj data-testid attribútumokat
2. Kerüld a fix várakozásokat (cy.wait(ms))
3. Szervezd modulárisan a tesztjeidet
4. Használd ki a session cache-t
5. Implementálj proper error handlinget
6. Automatizáld a CI/CD pipeline-ba

A Cypress elsajátítása időt igényel, de a befektetett energia megtérül a stabil és megbízható tesztekkel.