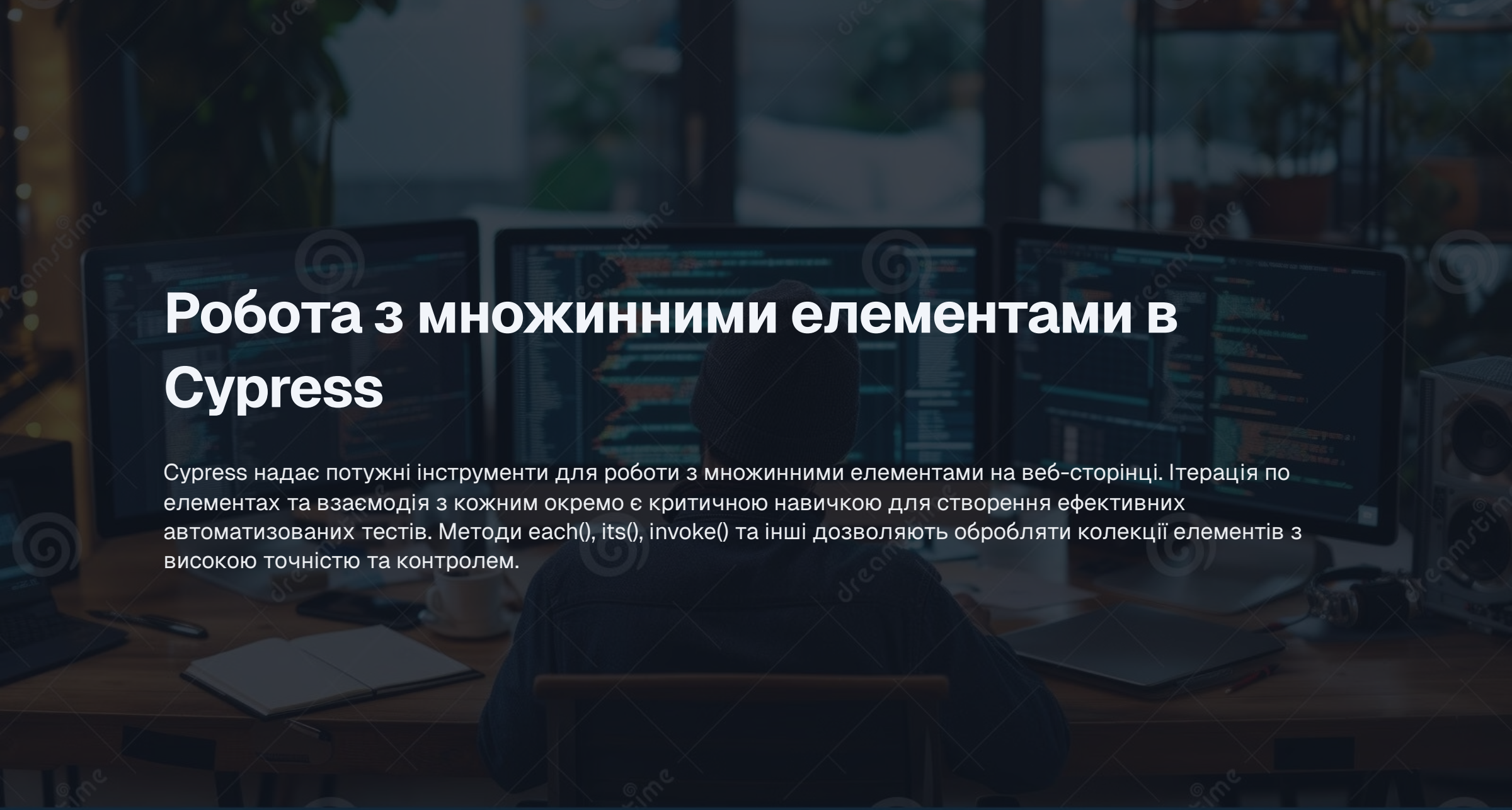


Робота з множинними елементами в Cypress

Cypress надає потужні інструменти для роботи з множинними елементами на веб-сторінці. Ітерація по елементах та взаємодія з кожним окремо є критичною навичкою для створення ефективних автоматизованих тестів. Методи each(), its(), invoke() та інші дозволяють обробляти колекції елементів з високою точністю та контролем.

Основні методи ітерації

each()

Ітерація по всіх елементах у виборі з виконанням дії для кожного

```
cy.get('.list-item').each(($item) => {
  cy.wrap($item).click();
});
```

its()

Витягування значень властивостей з об'єктів для подальшої роботи

```
cy.get('.list-item')
  .its('text')
  .should('include', 'Text');
```

invoke()

Виклик методів на кожному елементі у виборі

```
cy.get('.list-item')
  .invoke('text')
  .should('include', 'Text');
```

filter()

Вибір підмножини елементів за критерієм

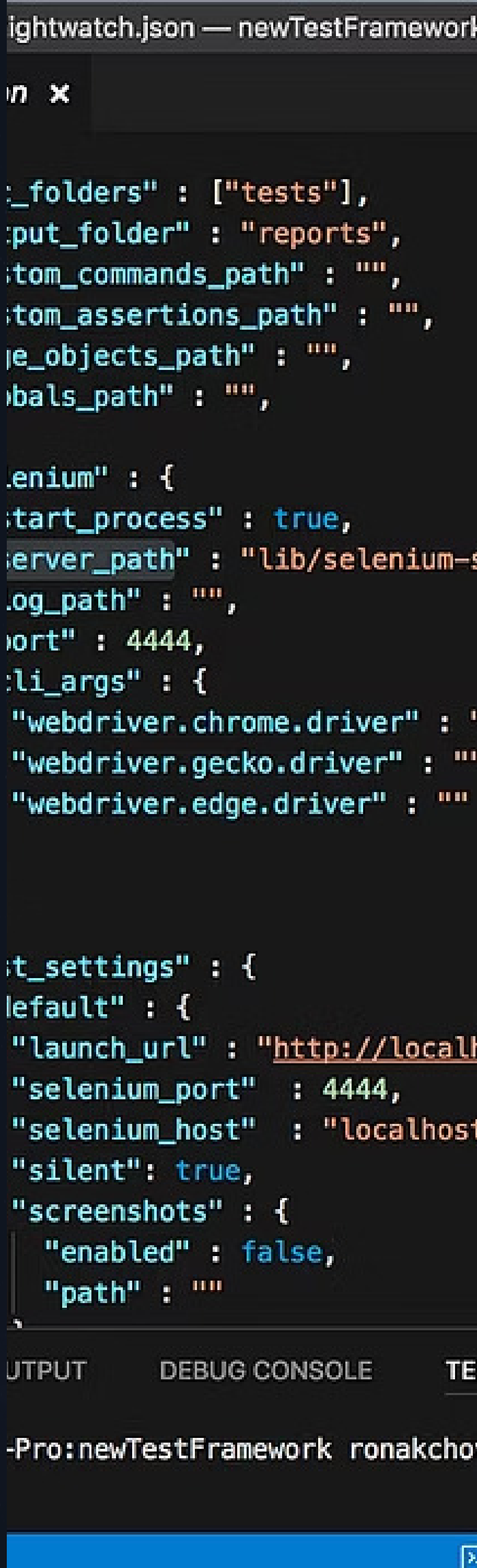
```
cy.get('.list-item')
  .filter(':contains("Text")')
  .should('have.length', 2);
```

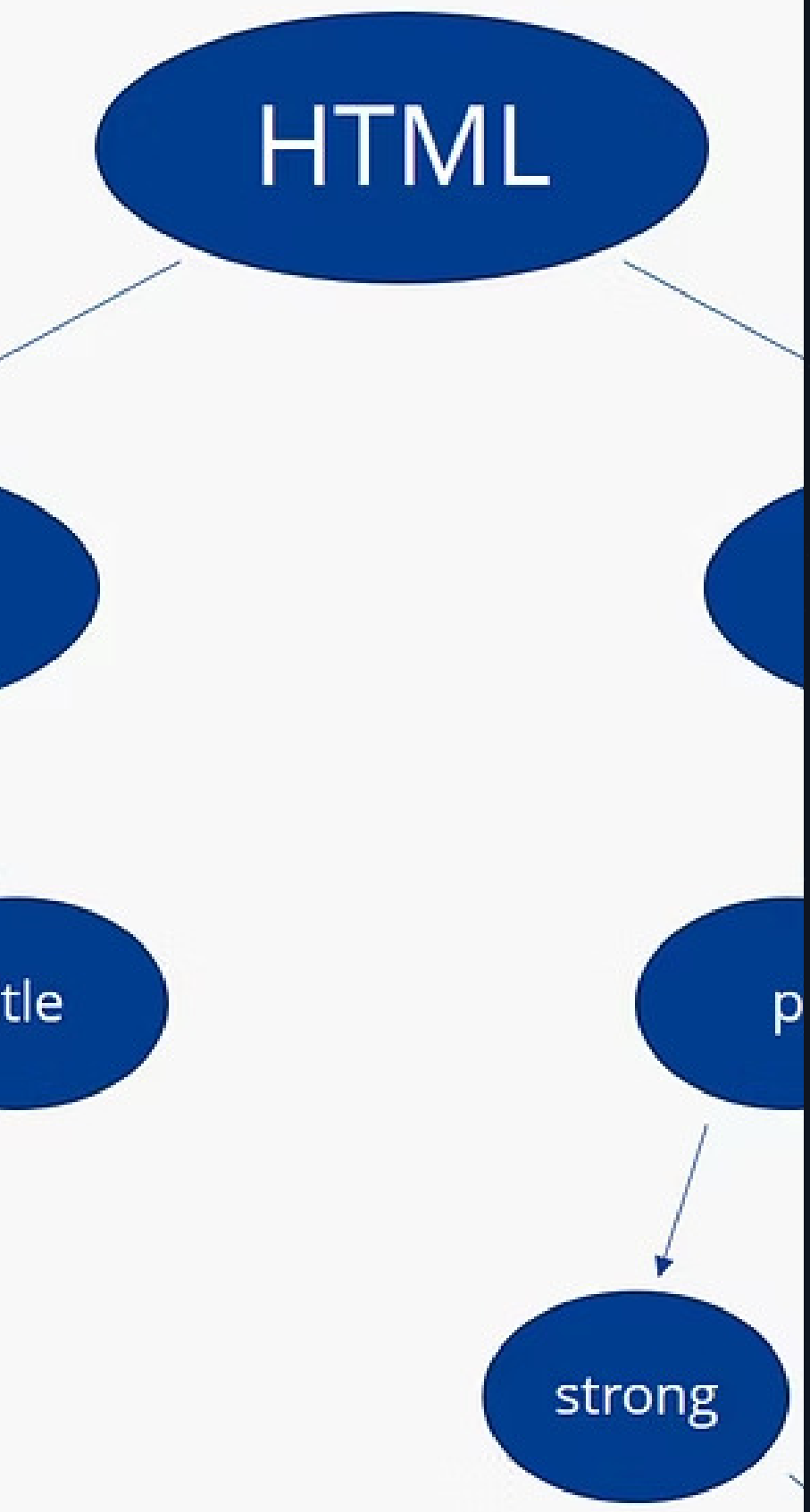
find()

Пошук вкладених елементів всередині кожного елемента

```
cy.get('.parent')
  .find('.child')
  .should('exist');
```

Комбінування методів дозволяє створювати складні сценарії тестування. Наприклад, each() для ітерації з подальшою взаємодією та перевіркою кожного елемента окремо. Вибір методу залежить від конкретних потреб тесту та структури DOM.





Селектори для навігації в DOM

first()

Обирає перший елемент у наборі

```
cy.get('nav a').first()
```

last()

Обирає останній елемент у наборі

```
cy.get('nav a').last()
```

nextAll()

Обирає всіх наступних "братніх" елементів

```
cy.get('.active').nextAll()
```

nextUntil()

Наступні елементи до певного селектора

```
cy.get('div').nextUntil('.warning')
```

Методи навігації по DOM дереву допомагають точно знаходити потрібні елементи відносно інших. Використання first() та last() спрощує роботу з колекціями, а nextAll() та nextUntil() дозволяють вибирати групи сусідніх елементів для масових операцій.

Робота з батьківськими та попередніми елементами

parents()

Обирає всі батьківські елементи знайденого раніше

```
cy.get('.child')
  .parents('.ancestor')
  .should('exist');
```

prevAll()

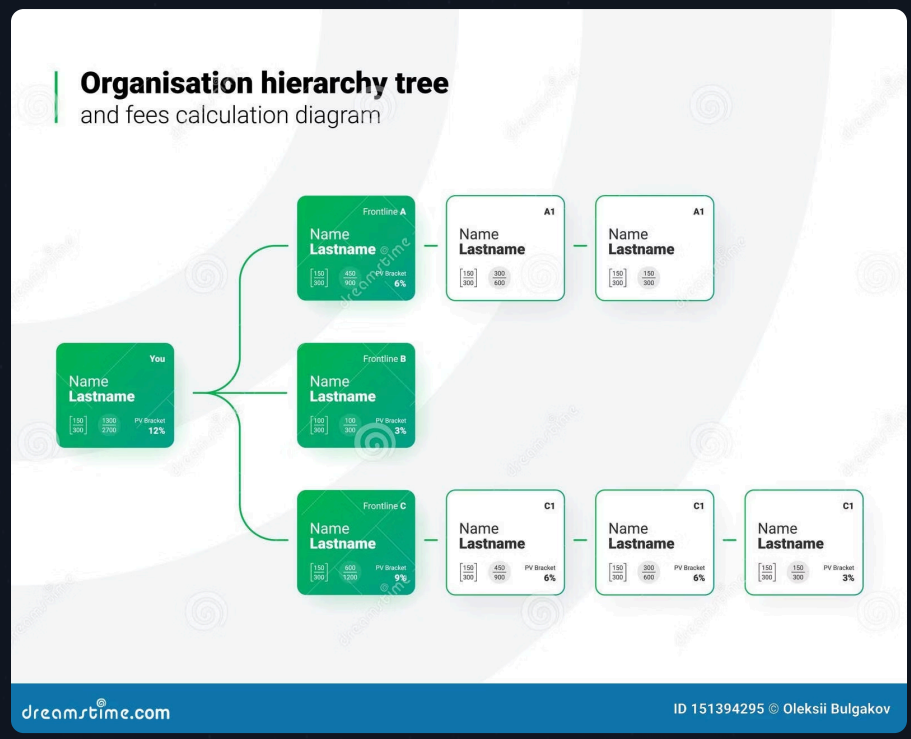
Обирає всі попередні "братні" елементи

```
cy.get('.active')
  .prevAll()
  .should('have.length', 3);
```

prevUntil()

Попередні елементи до певного селектора, не включаючи його

```
cy.get('p')
  .prevUntil('.intro')
  .should('be.visible');
```



Навігація вгору по DOM або назад серед "братніх" елементів розширює можливості тестування. Parents() знаходить всіх предків, що корисно для перевірки контексту елемента. PrevAll() та prevUntil() дзеркально відображають функціонал nextAll() та nextUntil(), дозволяючи рухатися в обох напрямках по DOM дереву.

Практичний приклад: робота з масивами

Робота з множинними елементами часто вимагає збору даних у масив для подальшого аналізу. Cypress дозволяє витягувати текст з кнопок, порівнювати масиви та виконувати складні перевірки.

```
cy.get('nb-card[size="small"]').first()
  .find('button.status-danger')
  .then(($elements) => {
    const buttonsText = []
    cy.wrap($elements).each(($el) => {
      const text = $el.text()
      buttonsText.push(text)
    }).then(() => {
      cy.log(`Buttons text: ${JSON.stringify(buttonsText)}`)
      expect(buttonsText).to.deep.equal(['Left','Top','Bottom','Right'])
    })
  })
```

01

Знаходження елементів

Отримання колекції кнопок за селектором

02

Ітерація та збір даних

Проходження по кожному елементу та збір тексту

03

Перевірка результату

Порівняння отриманого масиву з очікуваним

Цей підхід демонструє силу комбінування методів then(), wrap() та each() для складних операцій з даними. Логування результатів допомагає в дебагінгу, а deep.equal забезпечує точну перевірку вмісту масивів.

Best Practices: селектори та структура

1

Атрибути data-cy

Використовуйте фіксовані атрибути для тестових селекторів

```
<button data-
cy="submit">Відправити</button>
```

2

cy.contains() для тексту

Вибір елементів за текстовим вмістом

```
cy.contains('.menu li', 'Опція 3')
```

3

Аліаси елементів

Створюйте аліаси для покращення читабельності

```
cy.get('.username').as('usernameInput');
cy.get('@usernameInput').type('user123');
```

4

Розділення тестів

Окремі випадки для кращої підтримки

```
describe('Логін', () => {
  it('помилка при невірних даних', () => {});
  it('успішний вхід', () => {});
});
```

Використання правильних селекторів критично для стабільності тестів. Атрибути data-cy не змінюються при рефакторингу UI, що робить тести більш надійними. Аліаси спрощують код та підвищують його читабельність.

Додаткові Best Practices

find() для вкладених

Використовуйте для звернення до вкладених елементів всередині іншого

```
cy.get('.parent').find('.child').should('exist');
```

Змінні для селекторів

Зберігайте селектори у змінних для повторного використання

```
const submitButton = '.submit-button';
cy.get(submitButton).click();
```

within() для груп

Обмежує дії всередині обраного елемента

```
cy.get('.form').within(() => {
  cy.get('.username').type('user');
});
```

Обережно з force

Використовуйте .force() тільки коли це дійсно необхідно

Стабільні атрибути

Віддавайте перевагу атрибутам, які менше схильні до змін: data-testid, data-cy, id. Уникайте класів, які часто змінюються при стилізації.

first() та last()

Для швидкого доступу до першого або останнього елемента у колекції без необхідності додаткової логіки.

```
cy.get('.list-item').first().should('have.class', 'first');
```

Ці практики допомагають створювати чистий, підтримуваний код тестів. Within() особливо корисний для роботи з формами та складними компонентами, де потрібно обмежити область пошуку елементів. Змінні для селекторів полегшують оновлення тестів при зміні структури сторінки.