Leveraging App Actions and Page Objects

Marko Vajs
Software Development Engineer in Test

Module Overview

Explore benefits and drawbacks of Page Object Model (POM) and Application Actions

Define and use App Actions

Implement Page Objects

Page Object Model and App Actions

Page Object Model

Wrapper over a web page

Design pattern where web pages are represented as classes

Encapsulates the mechanics required to interact with the user interface

```
it('should open the Breithorn adventure', () => {
   cy.get('a[href="/adventure/1"]').click();
   cy.get('#title').should('have.text', 'Breithorn, Pennine Alps');
});
```

```
it('should open the Breithorn adventure', () => {
   homePage.clickMoreDetailsBtn(1)
       .getAdventureTitle().should('have.text', 'Breithorn, Pennine Alps');
});
```

```
import { AdventureDetailsPage } from "./adventure-details.page";

export class HomePage {

   visit(): HomePage {
       cy.visit('/');
       return this;
   }

   clickMoreDetailsBtn(adventureId: Number): AdventureDetailsPage {
       cy.get(`a[href="/adventure/${adventureId}"]`).click();
       return new AdventureDetailsPage;
   }
}
```

Page Object Model

Benefits

Better maintainability when UI changes occur

Re-using the same code across multiple tests

Tests are easier to read and follow

Drawbacks

Easy to violate a single responsibility principle

Additional time and effort are required for maintenance

App Actions

An approach where tests directly access the internal implementation of the application under test

Enable changing application's state without interacting with the application through the UI

```
export class FilterComponent {
    …
    constructor(private filterService: FilterService) {
        this.filterService.filterValueChange.subscribe(value => this.filterValue = value);

        if (window.Cypress) {
            window.FilterComponent = this;
        }
    }

    onChange(value: string): void {
        this.filterService.filterBy(value);
    }
    …
```

```
it('should display filter criteria', () => {
        cy.visit('/');
        cy.window().then((window: Window) => {
                cy.wrap(window).its('FilterComponent').invoke('onChange', 'Tara');
        …
});
```

App Actions

Benefits	Drawbacks
Tests become much faster	Fail to catch certain bugs
Tests become more focused	Easy to create unrealistic test scenarios
Tests are cleaner and easier to maintain	Knowledge of the front-end framework of the application is required (Angular)
Test stability is increased	

There is no strict guide on how to implement Application Actions.

Demo

Exposing Angular components to Cypress

Demo

Automate a scenario where the filter value is set by directly changing the application's state:

- Expose entry `AppComponent`
- Expose `FilterComponent`
- Set filter value directly
- Automate remaining steps through UI

Demo

Implementing Page Object Model

Module Summary

Module Summary

App Actions and Page Objects are not mutually exclusive

Page Object pattern promotes code reusability and keeps tests clean and readable

App Actions expose Cypress's superpowers by allowing you direct access to the internal implementation of your front-end application

Up Next:
Testing Advanced Scenarios