



# **Cypress.io**

## **Das Tool für die Testautomatisierung von JavaScript Frameworks**

Fanni Marosi

24.05.2022



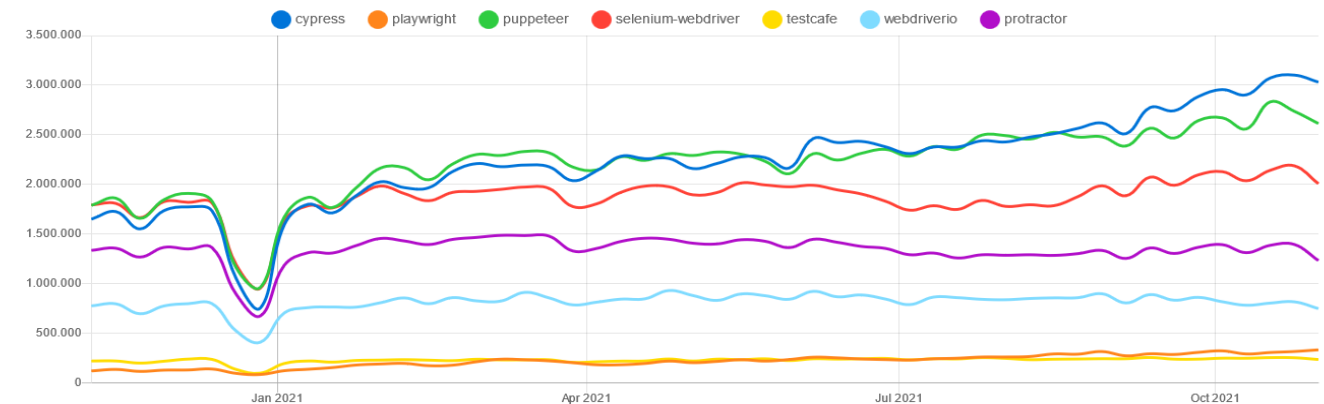
## Protractor Standard Testingframework -> 2021 April End-of-Life

In dem offiziellen GitHub Post von Protractor werden Alternativen vorgestellt.[1]

Cypress, PlayWright, Puppeteer, Selenium, TestCafe, WebdriverIO.

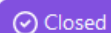
Indikatoren -> Npm Trends, Github

Downloads in past 1 Year



<https://www.npmtrends.com/cypress-vs-playwright-vs-puppeteer-vs-selenium-webdriver-vs-testcafe-vs-webdriverio-vs-protractor>

# Future of Angular E2E & Plans for Protractor #5502



Closed

kyliau opened this issue on 5 Apr 2021 · 80 comments



kyliau commented on 5 Apr 2021 · edited

Contributor



## TLDR

The Angular team plans to end development of Protractor at the end of 2022 (in conjunction with Angular v15).

Why?

Protractor was created in 2013 when WebDriver APIs were not yet a [standard](#) and end-to-end (e2e) tests were hard to write due to lack of support for `async / await`. To solve this problem, Protractor wraps [selenium-webdriver](#) and abstracted asynchronous operations from developers with the use of [Control Flow](#).

Since then, the JavaScript standard and ecosystem advanced considerably, providing modern syntax and much better development tools. Nonetheless, Protractor is not able to leverage such technology without forcing users to rewrite their tests. Meanwhile, robust alternatives have emerged in the web testing space. Developers will see more benefits from adopting a more modern testing tool than from updating to a breaking version of Protractor which does not provide additional functionality or developer ergonomic improvements.

We would like to hear from the community on

- the deprecation timeline
- what we can do to provide reliable integration with third-party solutions
- how users can transition by following migration guidelines
- additional concerns that would ensure a smooth transition

This RFC will close on Friday April 16, 2021.

Die Aufgabe der **Qualitätssicherung**, jedwede Art von Fehlern vor der Auslieferung einer Software zu finden und das Auftreten von Fehlern beim Kunden zu vermeiden.

Formulierung - „der Suche nach Fehlern“

Man sollte trotzdem im Hinterkopf haben, dass Tests dazu geschrieben werden, **um Fehlverhalten aufzudecken** und dass die Suche nach dem zugehörigen Fehler ein zweiter Schritt ist, der z. B. ein systematisches Vorgehen beim Debugging benötigt.

## Was ist ein Testfall?

das Ausprobieren einer Software

Drei Teilschritte:

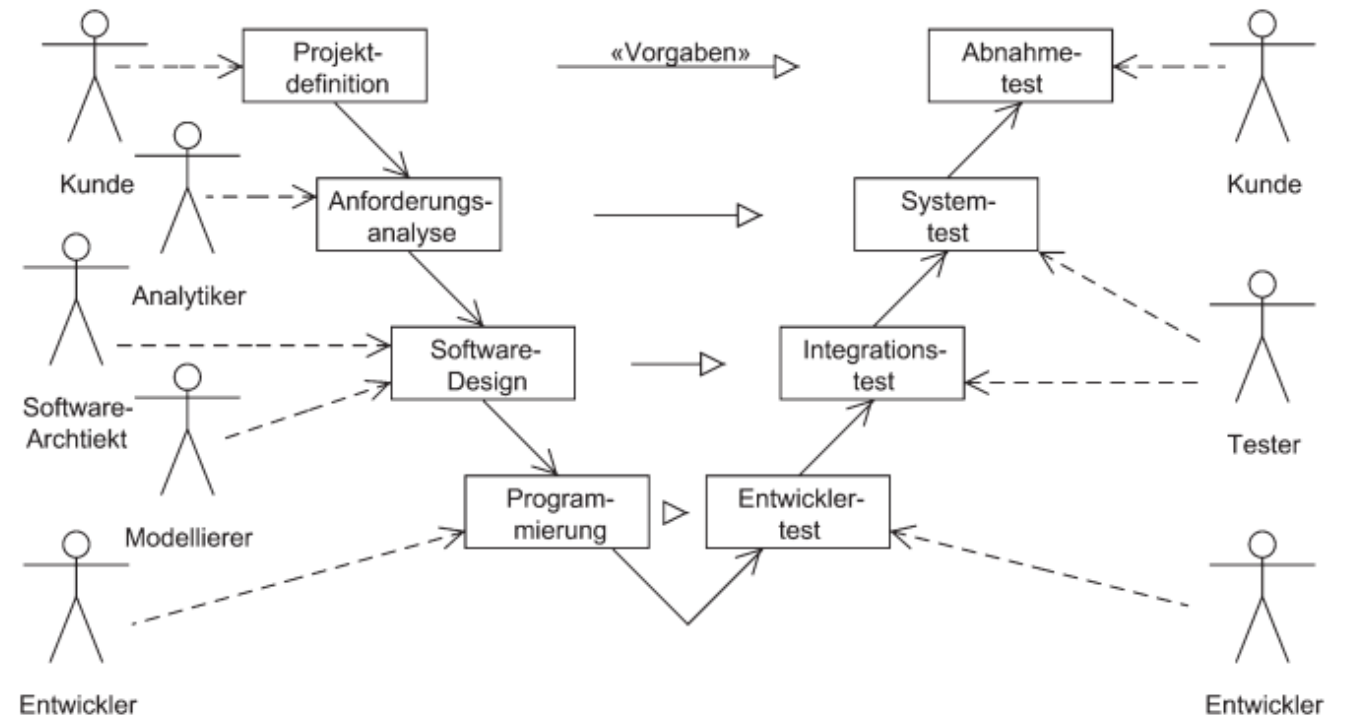
1. die Vorbedingungen,
2. die Ausführung
3. und die Nachbedingungen

Generelles Ziel der Testfallspezifikation muss es sein, den gleichen Test unter den gleichen Vorbedingungen immer wieder durchzuführen und dann zu den gleichen Ergebnissen zu kommen.

**Aus Anforderungen Testfälle erstellen :**

1. Szenariobeschreibung (Testgegenstand)
2. Berücksichtigung der zugrunde liegenden Anforderung (Abdeckung einer oder mehrerer Anforderungen mit entsprechenden Testschritten)
3. Erwartetes Ergebnis

Testphase	Teststart
Systemtest	Smoke-Test
	Integrationstest
	Regressionstest
Funktionstest	End-to-End-Test
	Operational Acceptance Test (OAT)
	UAT (User-Acceptance-Test)
	Go-Live-Test
Nicht-funktionale Tests	Performancetest
	Security-Test
	Disaster Recovery Test (DR Test)



Testarten

V-Modell mit Entwicklung und Test

Projektphase	Aktivität/Bericht	Inhalt/Darstellung
Initialisierung		
Konzept/Spezifikation	Testkonzept/Übersicht aller geplanten Berichte	<ul style="list-style-type: none"> <li>Aufführung und Abnahme der geplanten Berichte</li> </ul>
Realisierung	Testfallvorbereitung	<ul style="list-style-type: none"> <li>Anzahl der erstellten vs. geplanten Testfälle (absolut, prozentual, mit oder ohne Verlauf)</li> </ul>
Test	Testdurchführung	<ul style="list-style-type: none"> <li>Testfortschritt vs. geplanter Testfortschritt aggregiert pro Bereich/Teilprojekt/Arbeitspaket sowie für das Gesamtprojekt</li> <li>Auslastung pro Tester</li> </ul>
	Defect-Situation	<ul style="list-style-type: none"> <li>Anzahl der Defects mit dazugehörigen Schwereklassen aggregiert pro Bereich/Teilprojekt/Arbeitspaket sowie für das Gesamtprojekt</li> </ul>
Einführung	Testergebnisbericht	<ul style="list-style-type: none"> <li>Pro Bereich: Aufführung pro Teilprojekt mit Darstellung der lt. Testkonzept festgelegten Testausführungsgrade und Fehler-situation mit Darstellung der festgelegten Testendekriterien und möglicher offener Punkte sowie Aggregationsmöglichkeiten für das Gesamtprojekt</li> </ul>
Stabilisierung		
Abschluss	Testabschlussbericht	<ul style="list-style-type: none"> <li>Zusammenfassung der Testergebnisse auf Gesamtprojekt-Ebene</li> </ul>

## 3.1 IEEE 829 Standard for Software Test Documentation

Im IEEE 829 Standard for Software Test Documentation werden acht Dokumente beschrieben, die in drei Kategorien unterteilt sind:

- **Übersicht**

1. **Testkonzept** (test plan): Das Testkonzept bestimmt Abgrenzung, Vorgehensweise, Mittel und Ablaufplan der Testaktivitäten. Es bestimmt die Elemente und Produktfunktionen, die getestet werden sollen, die Testaufgaben, die durchgeführt werden müssen, das verantwortliche Personal für jede Aufgabe und das Risiko, das mit dem Konzept verbunden ist.

- **Test-Spezifikation** (test specification)

1. Die **Testentwurfsspezifikation** (test design specification) verfeinert die Beschreibung der Vorgehensweise für das Testen der Software. Sie identifiziert die Produktfunktionen, die von den Tests abgedeckt werden müssen. Sie beschreibt weiterhin die Testfälle und Testabläufe, die benötigt werden, um Tests zu bestehen und spezifiziert die Bestehens- oder Verfehlenskriterien der einzelnen Produktfunktionen.
2. Die **Testfallspezifikation** (test case specification) dokumentiert die zu benutzenden Eingabewerte und erwarteten Ausgabewerte. Testfälle sind vom Testdesign getrennt. Dies erlaubt die Verwendung der Testfälle in mehreren Designs und die Wiederverwendung in anderen Situationen.
3. Die **Testablaufspezifikation** (test procedure specification) ist die Beschreibung aller Schritte zur Durchführung der spezifizierten Testfälle und Implementierung des zugehörigen Testdesigns.

- **Testbericht** (test reporting)

1. Der **Testobjektübergabebericht** (test item transmittal report) beschreibt die Übergabe der Testfälle für den Fall, dass getrennte Entwicklungs- und Testteams eingebunden sind oder für den Fall, dass ein offizieller Zeitpunkt für den Beginn einer Testausführung erwünscht ist.
2. Das **Testprotokoll** (test log) dient zur Aufzeichnung der Ereignisse während einer Testausführung.
3. Der **Testabweichungsbericht** (test incident report) dokumentiert alle Ereignisse, die während einer Testausführung auftreten und weitere Nachprüfungen erfordern.
4. Der **Testabschlussbericht** (test summary report) fasst die Testaktivitäten zusammen, die mit einer oder mehreren Testentwurfsspezifikationen zusammenhängen.



**Tab. 17.1** Beschreibung eines Testfalls

Testfall	Eindeutige Identifikationsnummer
Testgegenstand/ Testobjekt:	Beschreibung der zu testenden Anwendung
Testkonfiguration/ Testvoraussetzung/ Testdaten:	Welche Systemumgebung und welche Umgebungsvoraussetzungen sind Voraussetzung für den Test? Verweis auf die zum Test notwendigen Testdaten und mögliche Randbedingungen
Testbeschreibung:	Genaue Beschreibung des Testfalls und der Schritte die bei der Ausführung zu beachten sind (Testprozeduren )
Bezug:	Bezug zum Pflichtenheft (z. B. Use Case „Datei Öffnen“ S. X). Möglichst viele Punkte aus dem Pflichtenheft durch Testfälle abdecken. Außerdem soll hier der Bezug zu anderen Tests vermerkt werden, insbesondere wenn es sich um eine Wiederholung eines Testfalles handelt.
Priorität:	Priorität des Testfalls (unbedingt erforderlich oder eher nachgeordnet?)
Details:	Evtl. weitere benötigte Details zum besseren Verständnis oder Ergänzungen. Evtl. auch angedeutete Lösungsvorschläge.
Soll-Ergebnis/ Test-Orakel/erwartetes Ergebnis:	Welches Ergebnis wird erwartet bzw. soll laut Spezifikation erzielt werden?
Ist-Ergebnis/ tatsächliches Ergebnis/ Testausgang:	Was ist das tatsächliche Ergebnis?
Bestanden:	War der Test erfolgreich?
Aus welcher Phase stammt der Fehler:	Falls ein Fehler vorliegt, soll hier aufgeführt werden in welcher Phase der Fehler eingebaut wurde (Grobentwurf, Feinentwurf/Design, Implementierung, Test).
Kommentar:	Kommentare und Anmerkungen zu den Ergebnissen.
Tester:	Wer hat den Test durchgeführt?
Datum/Uhrzeit:	Wann wurde der Test durchgeführt?

Welche **Aspekte** im Einzelnen gewählt werden und wie ausführlich die Testspezifikationen zu erstellen sind, ist **vom Testobjekt und von den Requirements** an das Endprodukt **abhängig**.

Der Test sollte immer so beschrieben werden, dass ihn auch jemand durchführen kann, der über keine Detailkenntnisse des zu testenden Systems verfügt.

Idealerweise sollte man **jemanden von der Straße holen** können, der ein grundlegendes Verständnis von Software hat, ihm die Testanweisung in die Hand drücken und er sollte dann in der Lage sein, ohne Rückfragen den Test eigenständig durchzuführen.



Testfall ID/Name	Schritte	Aktion	Eingabedaten	Erwartetes Ergebnis	Status (Bestanden / nicht bestanden)	Ausführungsdatum	Software Version	Priorität
1	1	Username Passwort	Xy xy			23.05.2022	1.0.2	1
	2	Button Click		Weiterleitung auf Dashboard		23.05.2022		
	2.1	Button Click	Falsche Eingabedaten	Modal/Toast		23.05.2022		
Satus	Bestanden/ nicht bestanden   durchgeführt von XY							

## Ablauf der Testautomatisierung

Im Allgemeinen wird eine **spezialisierte Entwicklungsumgebung für eine Skriptsprache** benutzt.

Das Tool greift auf Schnittstellen der zu testenden Anwendung zu. -> Komponenten der Benutzeroberfläche, Buttons, Textfelder, Tabellen usw.

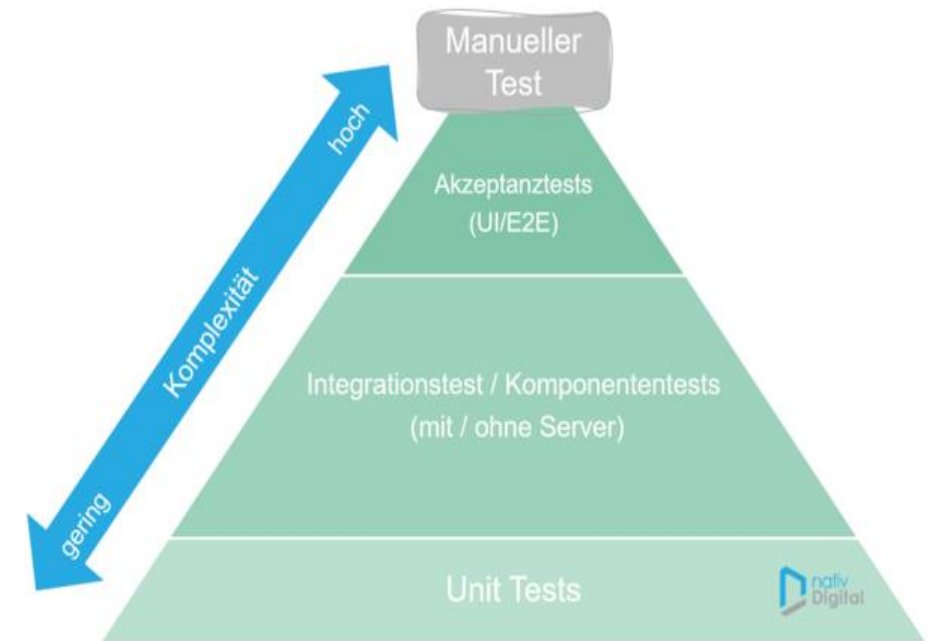
Skripte umsetzen

*„Tests, die wiederholt ablaufen, können automatisiert werden. Das verhilft zu höherer Testabdeckung, einer verlässlichen Reproduzierbarkeit, spart Zeit und Kosten. Wenn man 1000 Kombinationen von Testfällen manuell testen will, ist man damit ein paar Wochen beschäftigt. Ein Roboter der dafür ein Skript abarbeitet, schafft das in wenigen Stunden.“[2]*

**Akzeptanztests** -> gesamte Anwendung End-2-End getestet

Sind die Anforderungen erfüllt?

UI-basiert durchgeführt -> **reales Benutzerverhalten durch die Testskripte imitiert**



<https://nativdigital.com/testautomatisierung/>

Cypress-iFrame: Testcode und Anwendungscode im gleichen Browser-Tab ausgeführt

Cypress Core als Open Source und kostenlos  
(Dashboard Service als kommerzielles Produkt)

JavaScript

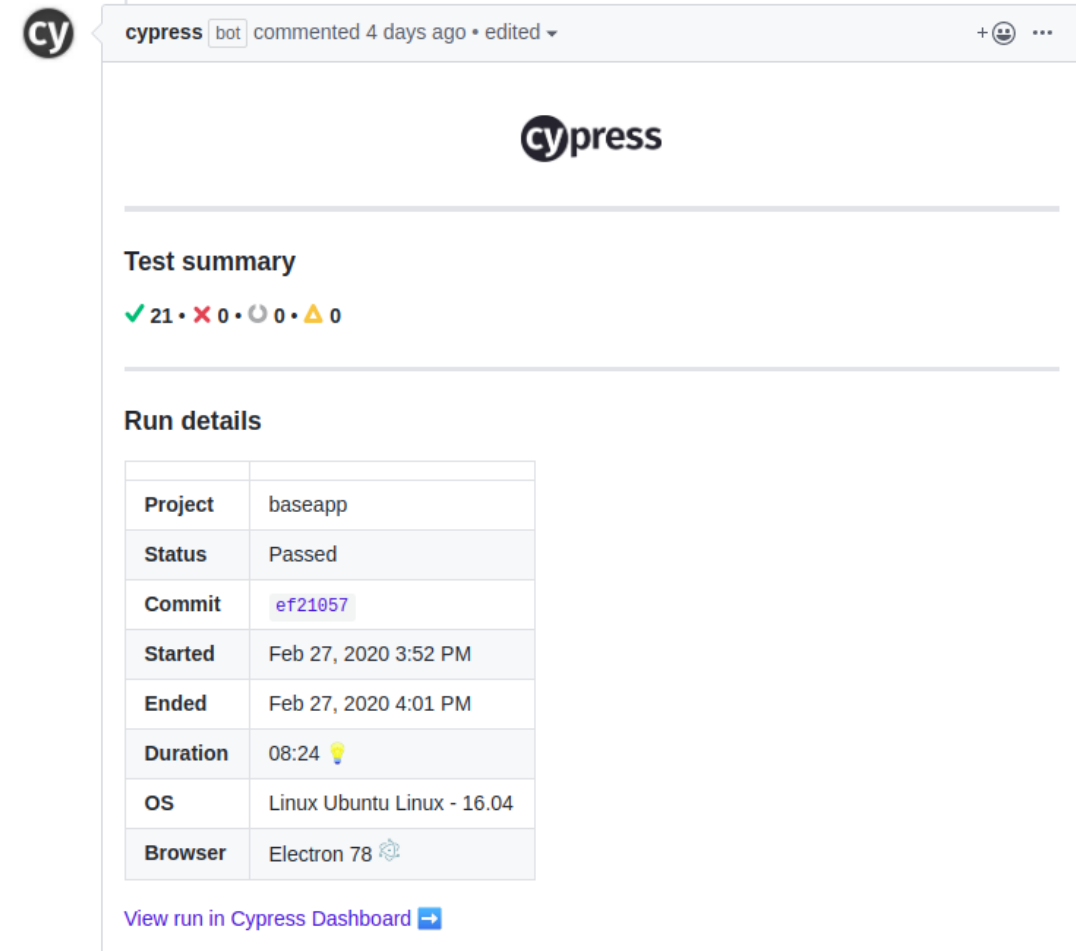
Chrome, Edge, Firefox, Electron

Mocha JS

Cy.wait() Automatisches Warten



- Zeit & Kosten reduzieren
- Cypress läuft direkt im Browser
- Out of the Box & Einfaches Setup -> npm-package
- Tests lassen sich einfach debuggen
- Die Unterstützung bei „wait of Element“ ist besser als Sel
- Videos lokal aufzeichnen
- CI Pipelines
- Plugins
- Kostenloser Mocha Reporter
- Docs, Support, Community



The screenshot shows a Cypress test run summary in a web browser. At the top, there's a header with the Cypress logo and a comment from 'cypress bot' stating 'commented 4 days ago • edited'. Below this, the Cypress logo is displayed again. The main content is divided into two sections: 'Test summary' and 'Run details'. The 'Test summary' section shows a status bar with 21 passed tests (green checkmarks), 0 failed tests (red X), 0 pending tests (grey circle), and 0 skipped tests (yellow triangle). The 'Run details' section contains a table with the following information:

Project	baseapp
Status	Passed
Commit	ef21057
Started	Feb 27, 2020 3:52 PM
Ended	Feb 27, 2020 4:01 PM
Duration	08:24 ⚡
OS	Linux Ubuntu Linux - 16.04
Browser	Electron 78 ⚙️

At the bottom of the 'Run details' section, there is a link that says 'View run in Cypress Dashboard' with an external link icon.

*`npx cypress open`*

*`npx cypress run`*

*`npx cypress run --reporter mochawesome \`  
`npx mochawesome-merge "cypress/results/*.json" >`  
`mochawesome.json`*

*`npx marge mochawesome.json`*

*Remove the mochawesome.json file  
from root directory bevor commit!*

Spec		Tests	Passing	Failing	Pending	Skipped
✓ test/1_browser_actions.spec.js	00:05	6	6	-	-	-
✓ test/2_buttons.spec.js	00:09	8	8	-	-	-
✓ test/3_inputs.spec.js	00:06	7	7	-	-	-
✓ test/4_select_box.spec.js	00:13	5	5	-	-	-
✓ test/5_fixture.spec.js	00:04	2	2	-	-	-
✓ test/6_actions.spec.js	00:18	14	14	-	-	-
✓ test/7_assertions.spec.js	00:04	9	9	-	-	-
✓ All specs passed!	01:01	51	51	-	-	-

The screenshot displays the Cypress test runner interface. The top bar shows the test name 'cypress-automation' and the timestamp 'Monday, May 23, 2022 11:48pm'. The left sidebar contains filters for 'Show Passed', 'Show Failed', 'Show Pending', and 'Show Skipped', all of which are currently active. Below these are sections for 'Show Hooks' (Failed), 'Browser actions', 'Working with inputs', 'Fixtures and Keyboard', 'Actions', 'Assertions', 'Implicit Assertions', 'Explicit Assertions', and 'Browser actions'. The main area shows two test suites: 'Browser actions' and 'Working with inputs'. Both suites show a list of test cases with their respective durations and status (all passed).

Test Suite	Test Case	Duration	Status
Browser actions	should load books url	2.3s	Passed
	should click on poetry category	1.1s	Passed
	should have correct price tag	79ms	Passed
	should have correct price tag	55ms	Passed
	should click on Home	548ms	Passed
	should click on Travel category	898ms	Passed
	should display correct number of books	59ms	Passed
	should take a snapshot	1.4s	Passed
Working with inputs	should load books url	947ms	Passed
	should fill username	276ms	Passed
	should fill password	1.7s	Passed
	should submit login form	377ms	Passed
	should display error	32ms	Passed
	should display error message	60ms	Passed
	should display error message	60ms	Passed



<http://books.toscrape.com/index.html>

## Testfall

1. Besuch die Seite
2. Bücherkategorie auswählen
3. Überprüfen ob .....

```
describe('Browser actions', () => {  
  //Load Url  
  it('should load books url', () => {  
    cy.visit('http://books.toscrape.com/index.html', { timeout: 10000 })  
    cy.url().should('include', 'index.html')  })  
  .  
  .  
  .  
  //Interaction with Buttons  
  it('should click on Travel category', () => {  
    //Targeting an html element  
    cy.get('a').contains('Travel').click()  
    cy.get('h1').contains('Travel')  
  })  
})
```

- ✓ should load books url
- ✓ should click on poetry category
- ✓ should have correct price tag
- ✓ should have correct price tag
- ✓ should click on Home
- ✓ should click on Travel category

## ▼ TEST BODY

```
1 get a 94
2 - contains Travel
3 - click
  (page load) --page loaded--
  (new url) http://books.toscraper.com/catalogue/cate...
4 get h1
  - contains Travel
```

## Books to Scrape We love being scraped!

Home / Books / Travel

## Books

- Travel
- Mystery
- Historical Fiction
- Sequential Art
- Classics
- Philosophy
- Humor
- Women's Fiction
- Fiction
- Childrens
- Religion
- Nonfiction
- Misc
- Default
- Science Fiction
- Sports and Games
- Add a category
- Fantasy
- New Adult
- Young Adult
- Science
- Poetry
- Paranormal
- Art
- Psychology
- Autobiography
- Parenting
- Adult Fiction
- Humor
- Horror
- History
- Food and Drink
- Christian Fiction
- Business
- Biography

## Travel

11 results.

Warning! This is a demo website for web scraping purposes. Prices and ratings here were randomly assigned and have no real meaning.



★★★★★

It's Only the Himalayas

£45.17

✓ In stock

Add to basket



★★★★★

Full Moon over Noah's Ark

£48.43

✓ In stock

Add to basket



★★★★★

See America: A Celebration ...

£48.87

✓ In stock

Add to basket



★★★★★

Vegetarianism: An Uncommon Guide ...

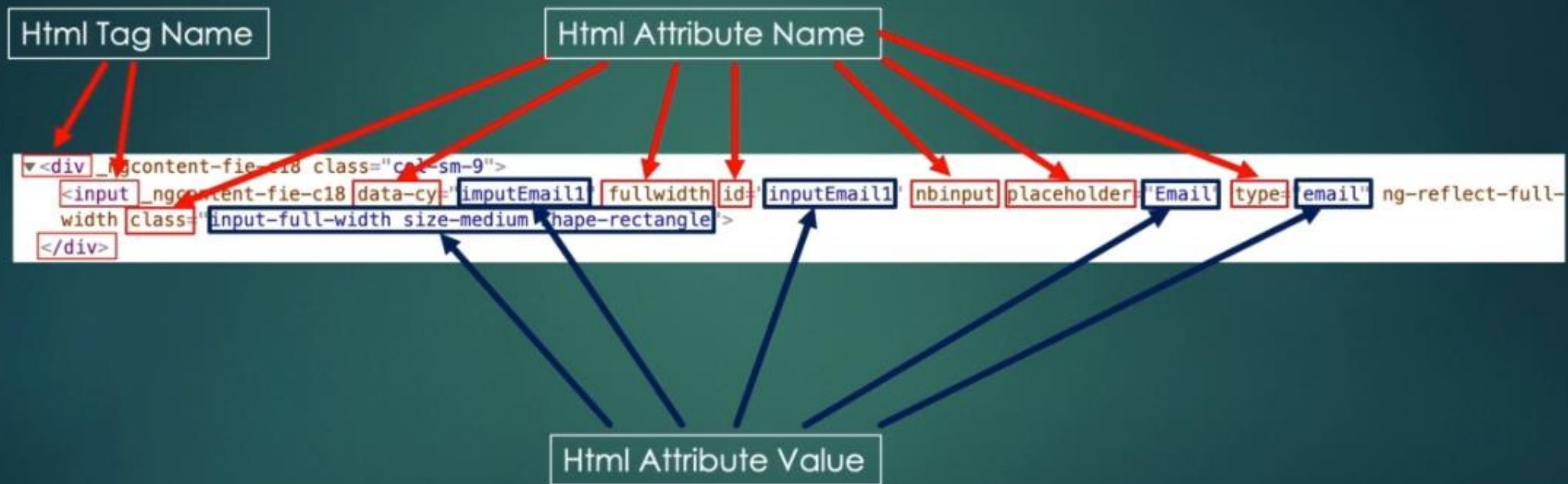
£38.94

✓ In stock

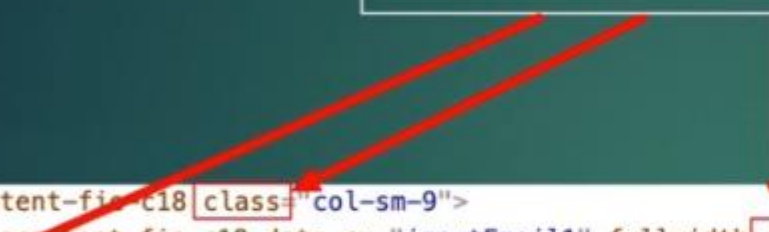
Add to basket




## Understanding DOM and terminology



"Class" and "ID" are also Attribute Names



```
▼<div _ngcontent-fie-c18 class="col-sm-9">  
  <input _ngcontent-fie-c18 data-cy="inputEmail1" fullwidth id="inputEmail1" nbinput placeholder="Email" type="email" ng-reflect-full-  
width class="input-full-width size-medium shape-rectangle">  
</div>
```



```
▼<div _ngcontent-fie-c18 class="col-sm-9">  
  <input _ngcontent-fie-c18 data-cy="inputEmail1" fullwidth id="inputEmail1" nbinput placeholder="Email" type="email" ng-reflect-full-  
width class="input-full-width size-medium shape-rectangle">  
</div>
```

Class value 1

Class value 2

Class value 3

The diagram shows a snippet of HTML code with several annotations:

- Opening Tag:** Red arrows point to the opening tags `<tbody>`, `<tr>`, and `<td>`.
- Closing Tag:** Blue arrows point to the closing tags `</tbody>`, `</tr>`, and `</td>`.
- Nested Content:** A yellow arrow points to the text `1.8 MB` inside a `<td>` tag.
- Text Value:** A blue arrow points to the text `5` inside a `<td>` tag.

```
<tbody role="rowgroup">
  <tr _ngcontent-ero-c27 class="nb-tree-grid-row ng-star-inserted" nbtreegridrow role="row">
    <td _ngcontent-ero-c27 class="nb-tree-grid-cell cdk-column-name nb-column-name ng-star-inserted" nbtreegridcell role="gridcell">...</td>
    <td _ngcontent-ero-c27 class="nb-tree-grid-cell cdk-column-size nb-column-size ng-star-inserted" nbtreegridcell role="gridcell">1.8 MB</td>
    <td _ngcontent-ero-c27 class="nb-tree-grid-cell cdk-column-kind nb-column-kind ng-star-inserted" nbtreegridcell role="gridcell">dir</td>
    <td _ngcontent-ero-c27 class="nb-tree-grid-cell cdk-column-items nb-column-items ng-star-inserted" nbtreegridcell role="gridcell">5</td>
  </tr>
  <tr _ngcontent-ero-c27 class="nb-tree-grid-row ng-star-inserted" nbtreegridrow role="row">...</tr>
  <tr _ngcontent-ero-c27 class="nb-tree-grid-row ng-star-inserted" nbtreegridrow role="row">...</tr>
</tbody>
```





```
6
7 //by Tag Name
8 cy.get('input')
9
10 //by ID
11 cy.get('#inputEmail1')
12
13 //by Class name
14 cy.get('.input-full-width')
15
16 //by Attribute name
17 cy.get('[placeholder]')
18
19 //by Attribute name and value
20 cy.get('[placeholder="Email"]')
21
22 //by Class value
23 cy.get('[class="input-full-width size-medium shape-rectangle"]')
24
25 //by Tag name and Attribute with value
26 cy.get('input[placeholder="Email"]')
27
28 //by two different attributes
29 cy.get('[placeholder="Email"][type="email"]')
30
31 //by tag name, Attribute with value, ID and Class name
32 cy.get('input[placeholder="Email"]#inputEmail1.input-full-width')
33
34 //The most recommended way by Cypress
35 cy.get('[data-cy="inputEmail1"]')
36 })
37
```



# Cypress.io



iFrame Support

Probleme mit Domainnamen -> Anmeldung druch SSO

Viel Workarounds

Danke für  
die Aufmerksamkeit!

Fragen?