

**cypress**

# ORGANISATORISCHES

- Mittagspause 12:00 bis 13:00
- Zu jeder vollen Stunde 10 Minuten Pause

# ÜBER MICH - JOSEF BIEHLER

- Wirtschaftsinformatik Bachelor
- Angestellt bei Samhammer AG,  
Weiden
- Fullstack Entwickler
- .NET Backend mit Aurelia Frontend
- UI Tests / API Tests
- Zeichnen Programmieren Bloggen



# SOCIAL MEDIA

- <https://biehler-josef.de>
- <https://kack.dev>
- <https://www.instagram.com/josefbiehler/>
- <https://twitter.com/JosefBiehler>
- <https://twitter.com/KackDev>
- <https://github.com/gabbersepp>
- <https://dev.to/gabbersenn>

# CODE

- <https://github.com/gabbersepp/cypress-workshop>

# ABLAUF

- Cypress zeigen mit einfachen Beispielen
- Dann: Umsteigen auf unsere Applikation

# START

```
npm init  
npm install cypress@7.3.0
```

## Zwei NPM Scripts anlegen:

```
"scripts": {  
  "cy-open": "cypress open",  
  "cy-run": "cypress run",  
  "cy-run-single": "cypress run --spec cypress/integration/t  
}
```

# KONFIGURATION

*cypress.json* im Rootverzeichnis anlegen:

```
{  
  "baseUrl": "http://biehler-josef.de:9500/HD",  
  "defaultCommandTimeout": 30000  
}
```

# VERZEICHNISSE

- */cypress/integration*
- */cypress/plugins*
- */cypress/support*



# STARTPROJEKT

<https://github.com/gabbersepp/cypress-workshop/tree/master/startproject>

- *npm install* nicht vergessen

# VS CODE TEMPLATES

- Gerade für Neueinsteiger im Team
- Hürden abbauen

# PLUGINS

- Ausführung im Node Prozess

```
module.exports = (on, config) => {  
}
```

# SUPPORT

- Einmalig ausgeführt in Browser
- Globale Konfig, Commands, etc

```
// leer
```

# ERSTER TEST

```
describe("erster test", () => {  
  it("blabla", () => {  
    cy.visit("https://samhammer.de");  
    cy.get("body").should("contain", "Samhammer");  
  })  
})
```

# ERSTER START

```
npm run cy-open
```

# INTERAKTIVER MODUS

- Snapshot pro Command
- kann jederzeit betrachtet werden
- Achtung: Speicherhungrig!

# ERSTER START

```
npm run cy-run
```



# TIPP: CYPRESS CACHE AUFRÄUMEN

- jede Cypress Version benötigt etwa 600MB
- C:/Users/jbiehler/AppData/Local/Cypress/Cache
- `cypress cache prune`
- oder einfach Ordner löschen ;-P

# ASSERTIONS

- Chai assertions
- BDD & TDD Syntax

```
// BDD:  
expect(name).to.not.equal('Jane')  
expect(arr).to.have.any.keys('age')  
expect(5).to.be.lessThan(10)  
  
// TDD:  
assert.notEqual(3, 4, 'vals not equal')
```

# ASSERTIONS - SHOULD

- Cypress Command um Assertions auszuführen
- Should + Chai Assertion möglich
- Should aber auch mit Callback

# EINSCHUB - CYPRESS = ASYNCHRON

- Cypress nutzt Promiseähnliches Konstrukt
- Cypress Commands geben "Chainable" Objekt zurück
- Ergebnis von `cy.get(...)` kann also nicht direkt verarbeitet werden, sondern nur mit weiteren Commands

# ASSERTIONS - SHOULD MIT CHAI ASSERT

- Assert auf das vorherige Element
- hier kann exakt der Gleiche Chai Assert benutzt werden

```
cy.get("body").should("contain", "Test")
```

# ASSERTIONS - SHOULD CALLBACK

```
cy.get(..).should($e => {  
  // Logik und Asserts  
})
```

# ASSERTIONS

- Kombination mit "and"

```
describe("assertions", () => {  
  it("test", () => {  
    cy.wrap({ test: 1 }).should("have.any.keys", "test")  
      .and("not.have.any.keys", "blabla")  
  })  
});
```

# ASSERTIONS

Cypresscommands haben Defaultassertions

```
cy.get("body123") // impliziert: should("exist")
```



# ASSERTIONS

Default assertions werden übersprungen

```
cy.get("body123").should("not.exist")
```

# NOT.VISIBLE VS NOT.EXISTS

```
cy.get("nicht-vorhanden").should("not.visible")
```

- führt zu Fehler
- nicht vorhandene Elemente können nur mit "not.exists" geprüft werden

# WEITERE SELEKTORFUNKTIONEN

- find()

```
cy.get("body").find("div")
```

- last()

```
cy.get("li").last()
```

- first()

```
cy.get("li").first()
```

# INTERAKTION

- type()

```
cy.get("input").type("bla{del}bl
```

- click()

```
cy.get("button").click()
```

- focus()

```
cy.get("input").focus()
```

# SELEKTOREN BESTIMMEN

- manchmal schwierig, wenn Elemente Fokus verlieren
- da hilft: interaktive Modus und DOM Snapshots!

**DEMO**

# ÜBUNG 1

- <https://docs.cypress.io/> aufrufen
- nach "cy.type" suchen
- Suchergebnis anklicken

# ÜBUNG 2

- <https://docs.cypress.io/> aufrufen
- Prüfen ob Text von jedem H3 größer 0 ist

# LÖSUNG 1

```
it("cy.type", () => {  
  cy.visit("https://docs.cypress.io/")  
  cy.get('.DocSearch-Button-Placeholder').click();  
  cy.get('#docsearch-input').type('cy.type').wait(1000);  
  cy.get('.DocSearch-Hit').first().click();  
})
```



# LÖSUNG 2

```
it("h3", () => {  
  cy.visit("https://docs.cypress.io");  
  
  cy.get("h3").then($divs => {  
    for(let i = 0; i < $divs.length; i++) {  
      expect($divs[i].textContent.length).greaterThan(0)  
    }  
  })  
})
```

# RETRYABILITY

- Manche Commands sind wiederholbar
- Es wird solange wiederholt bis Check erfolgreich oder Timeout abgelaufen ist

```
it("get() und should() wird wiederholt", () => {
  setTimeout(() => {
    const div = document.createElement("div");
    div.id="newdiv";
    div.textContent = "ASD";
    Cypress.$("body").append(div);
  }, 5000);

  cy.get("#newdiv").should($e => {
    console.log(`Length: ${$e.length}`);
    expect($e.length).greaterThan(0);
  });
})
```

▼ test

✓ get() und should() wird wiederholt

▼ TEST BODY

```
1 get      #newdiv
2 - assert expected 1 to be above 0
```

ASD

DevTools - biehler-josef.de:9500/\_/

Chrome wird von automatisierter Testsoftware gesteuert.

# THEN VS. SHOULD

- should: retryable
- then: nicht



▼ test

✖ nicht alles ist repeatable



▼ TEST BODY

1	get	body
2	- assert	expected 1 to equal 2

❗ AssertionError

expected 1 to equal 2

cypress/integration/3.retryability.spec.js:7:26

```
5 |         cy.get("body").then($e => {  
6 |             console.log(i++);  
> 7 |             expect(i).to.eq(2);  
   |                               ^
```

# SHOULD + COMMAND = VORSICHT

```
it("should + command = vorsicht", () => {  
  cy.get("body").should($body => {  
    cy.get("body")  
  })  
})
```

174	get	body
175	get	body
176	get	body
177	get	body
178	get	body
179	get	body
180	get	body
181	get	body
182	get	body
183	get	body
184	get	body
185	get	body
186	get	body
187	get	body

# VIDEOS UND SCREENSHOTS

Cypress macht automatisch Videos & Screenshots



# EIGENE SCREENSHOTS

- mit `cy.screenshot` command
- auch auf einzelne Elemente möglich

```
cy.get("button").screenshot()
```

# EIGENE SCREENSHOTS

- ermöglichen Visuelle Tests
- !! Screenshots unterscheiden sich (Environment, run, open)

# DEBUGGEN

- Am besten per `debugger` Statement

# AUF PROMISES WIRD GEWARTET

```
it("Auf Promises wird gewartet", () => {
  const promise = new Promise(resolve =>
    setTimeout(() => resolve("fertig"), 5000))

  cy.then({ timeout: 10000 }, () => promise)
    .then(result => expect(result).to.eq("fertig"))
})
// geht auch mit wrap:
const promise = new Promise(resolve => setTimeout(() => resolve(
  fn: () => "test"
), 5000))

cy.wrap(promise).invoke("fn").should("contain", "test")
```

# ASYNCHRONITÄT

- Cypresscommands landen in Queue
- Folge: Asynchronität

```
// falsch:
let $element = null;
cy.get("body").then($e => $element = $e);
expect($element).not.null;

// richtig:
let $element = null;
cy.get(".add-tab-btn")
  .then($e => $element = $e)
  .then(() => expect($element).not.null)
```

# ASYNCRONITÄT => KEIN AWAIT MÖGLICH

```
const body = await cy.get("body");
const text = body.text();
// "funktioniert"

expect(text).contain("Google");

// aber was ist damit:
cy.get("form[action*='search'] input[type='text']")
  .type("biehler-josef.de");
const result = await cy
  .get("[class*='__suggestions-inner-container']");

// schlägt fehl
expect(result.length).greaterThan(0);
```

**DEMO**

# DOKU

<https://docs.cypress.io/> Sehr ausführlich

# BEISPIELE

<https://github.com/cypress-io/cypress-example-recipes>

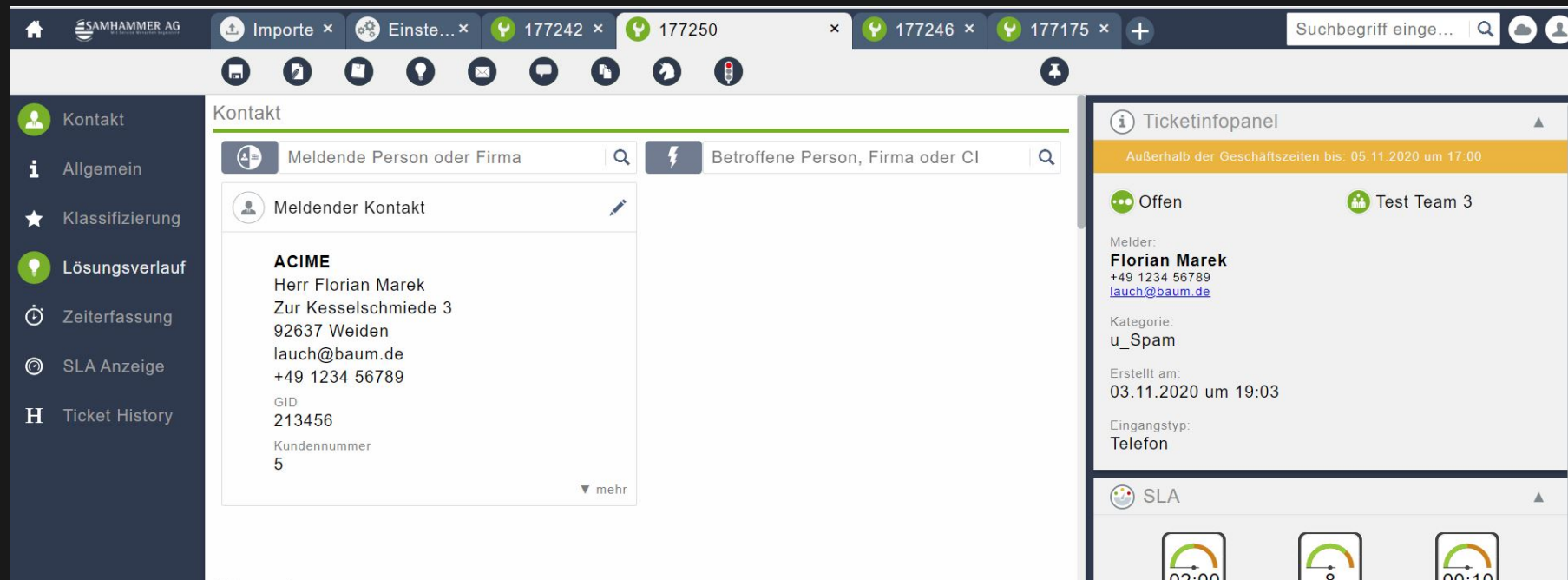
# BLOG

<https://www.cypress.io/blog/>  
<https://glebbahmutov.com/blog/>



# FIRSTANSWER HD

- Aurelia Frontend
- .NET Backend



# LOGIN

- in HD: Speicherung des Access Tokens in localStorage
- möglich über die GUI

```
cy.visit("/#/login");  
cy.get("#usernameinput").type("developer");  
cy.get("#passwordinput").type("Test12");  
cy.get("#loginbutton").click();  
cy.get(".add-tab-btn").should("exist");
```

Aber nicht sinnvoll

- man würde immer etwas testen, was man gar nicht testen will

# LOGIN

- besser mittels Request und Manipulation  
localStorage
- Code ist bereits im Startprojekt hinterlegt: *cy.login()*

# CY.REQUEST VS FETCH

- `cy.request` der natürliche Weg
- Aber: `fetch()` für generische Lösung

**DEMO**

# LOGIN ALS CUSTOM COMMAND?

- erweitert Cypress
- kann auf Chainable Objekte aufgerufen werden
- oder nur auf das globale Cy Objekt
- sinnvoll, wenn oft aufgerufen

```
// parent command
Cypress.Commands.add("login", (email, password) => { ... })
// child command --
Cypress.Commands.add("drag", { prevSubject: 'element' }, (subje
```

# CUSTOM COMMAND

## REPEATABLE?

- Beispiel: Command das eine Zufallszahl prüft
- Nützlich: `cy.verifyUpcomingAssertions`

# LOGGING

- `cy.log()`
- `Cypress.log`



# LOGGING IN CUSTOM COMMANDS

```
Cypress.log({  
  name: 'Name des Logs',  
  displayName: 'Anzeige in UI',  
  
  message: `Nachricht`,  
  consoleProps: () => {  
    return {  
      // das wird angezeigt, wenn man auf den Log klickt  
    }  
  }  
})
```

# CYPRESS.LOG VS CY.LOG

- Cypress.log ist kein asynchroner Command
- cy.log() schon
- cy.log() für "normale" Verwendungen
- Cypress.log() wenn keine asynchrone Vorgänge erlaubt sind

# FALSCH

```
it("test", () => {  
  cy.wrap({ test: 1 }).should(obj => {  
    cy.log("test")  
    expect(obj.test).eq(1)  
  })  
})
```

✖ test

▼ TEST BODY

wrap	{test: 1}
2 log	test
3 should	function(){} <b>TypeError</b> Cannot read property 'test' of null

cypress/integration/28.cy.log.falsch.spec.js:5:  
24

```
3 |         cy.wrap({ test: 1 }).should(o  
4 |             cy.log("test")  
> 5 |             expect(obj.test).eq(1)
```

# RICHTIG

```
it("test2", () => {  
  cy.wrap({ test: 1 }).should(obj => {  
    Cypress.log({  
      name: 'test',  
      displayName: 'test',  
      message: `test`,  
      consoleProps: () => {  
        return {  
          test: 1  
        }  
      }  
    })  
    expect(obj.test).eq(1)  
  })  
})
```

# CYPRESS LOGGING FORMAT

```
cy.log("test")  
cy.log("**test**")  
cy.log("_test_")  
cy.log("[red]test")
```

# DEBUG & PAUSE

- manchmal ganz hilfreich

# CHILD COMMAND

```
Cypress.Commands.add("doesContain", { prevSubject: "element" }  
  cy.wrap(subject).should("contain", options)  
  
  })  
  
it("test", () => {  
  cy.visit("https://google.de").get("body")  
    .doesContain("google");  
})
```

**DEMO**



# ÜBUNG

- HD aufrufen
- Element "#usernameinput" anfordern
- should + callback nutzen
- prüfen, ob Element vorhanden ist
- nein -> Exception werfen, ja -> weiter im Text
- Element loggen
- prüfen, ob element attribut 'type' = 'text'

# LÖSUNG

```
1
2 describe("HD", () => {
3     it("should be loaded", () => {
4         cy.visit("/");
5         cy.get("#usernameinput").should($e => {
6             if ($e.length === 0) {
7                 Cypress.log({
8                     name: 'Element noch nicht da',
9                     displayName: 'Element noch nicht da',
10                    message: `Element noch nicht da`,
11                });
12                throw new Error();
13            }
14
15            Cypress.log({
```

# PROBLEM: WELCHER TAB IST OFFEN?

- Person, Company, CI, Ticket ??
- und v.A. welche Id ??
- wichtig bei: Klick auf beliebiges Ticket in Pool etc
- immer dann, wenn keine spezifische Entität geöffnet wird

# PROBLEM: WELCHER TAB IST OFFEN?

Was wir brauchen:

- `cy.location()` zum Auslesen der ID
- Locator für Element, um gewechselten Tab sicherzustellen

# ÜBUNG

- Parent command für "Neues Ticket" welches die ID bereit stellt

## TIPPS

- ID von Button: #shellnewticket

```
Cypress.Commands.add(.....);
```



# ACHTUNG

- Location abgleichen!
- should() kann den returnvalue nicht manipulieren

# LOCATOR

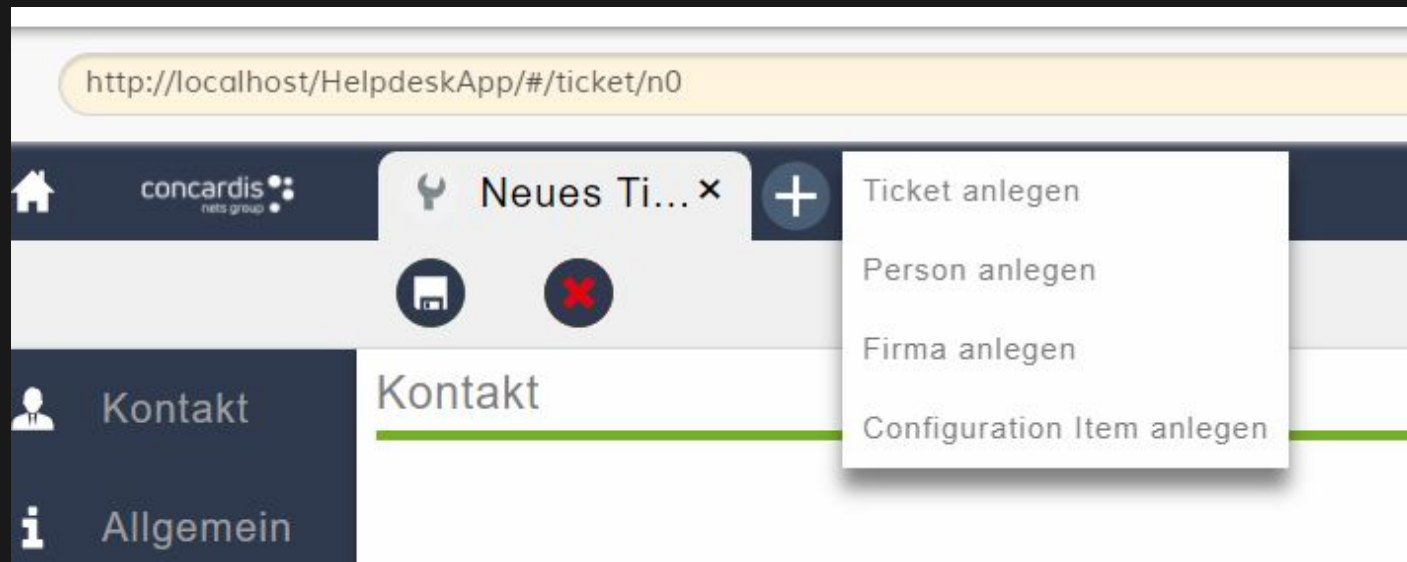
Nun können wir folgendes machen:

```
1
2 it("test2", () => {
3     cy.login("developer", "test");
4     cy.newTicket().then(id => {
5         cy.get(`#Title${id} input`).type("test");
6     })
7 })
```

Damit können wir mehrere Tickettabs geschickt ansprechen



# FLYOUT NACH "NEUES TICKET"



# STÖRENDE ELEMENTE

- entfernen von unerwünschten Flyout
- entfernen von Cookiebannern
- entfernen von Fixed Elementen
- .....

```
cy.get("body").then($e => $e.remove());  
// oder  
cy.get("body").click();
```

# PAGE OBJECT PATTERN

- Locator immer so zu schreiben ist nervig und Fehleranfällig
- unsere Lösung: Page Object (mehr oder weniger)

TS TicketPage.ts •

src > uitests > cypress > support > pages > TicketPage

```
51 public titleComponent: InputStringComponent =  
52     DependencyResolver.resolve<InputStringComponent>(InputStringComponent);  
53 |  
54 public descriptionComponent: InputRichEditorComponent =  
55     DependencyResolver.resolve<InputRichEditorComponent>(InputRichEditorComponent);  
56 |  
57 public categoryComponent: InputCategoryComponent =  
58     DependencyResolver.resolve<InputCategoryComponent>(InputCategoryComponent);  
59 |
```

```
protected prepareComponents(idSuffix: string): void {  
    this.idSuffix = idSuffix;  
    this.containerLocator = `#ticketBaseRoot${this.idSuffix}`;
```

```
    this.titleComponent.init({  
        containerLocator: this.containerLocator
```

```
export default class InputStringComponent extends BaseComponent {  
    private static InputLocator: string = "input";  
  
    public type(text: string, force: boolean = false): void {  
        cy.get(`${this.containerLocator} ${this.componentLocator}`  
            .type(text, { force: force }));  
    }  
  
    public clear(): Cypress.Chainable<JQuery<HTMLElement>> {  
        return cy.get(`${this.containerLocator} ${this.componentLocator}`  
            .clear();  
    }  
  
    public setValue(v: string, force: boolean = false): void {  
        this.type(v, force);  
    }  
  
    public getValue(): Cypress.Chainable<string> {  
        return cy.get(`${this.containerLocator} ${this.componentLocator}`  
            .val();  
    }  
}
```

# JETZT WIRD ES HÄSSLICH - SELECT2

- Auswahl von Kategorie

```
cy.newTicket();  
cy.get("#Categoryn0").click();  
// geht nicht :-(
```

# UMGANG MIT SELECT2

- Manche Libs machen die Testinteraktion schwierig
- siehe Code...

# PROBLEME ZWISCHEN TESTS INNERHALB EINER DATEI


- Cypress bereinigt State zwischen Tests
- aber kein Reset der App
- kein Reset von laufenden Requests!





cypress/integration/26.spec.js

▼ test

- ✓ Cypress bereinigt state zwischen tests
- ✓ deshalb ist der localStorage leer
- ✓ aber Cypress bereinigt nicht die App
- ✓ wie man sieht
- ✓ das führt zu problemen -> nur schnell die App öffnen
- ✗ und nochmal 

▼ TEST BODY

(xhr) http://localhost:3000/api/teams/GetTeamsSel...

(xhr) GET 401 /api/teams/GetTeamsSel...

# PROBLEME ZWISCHEN TESTS - LÖSUNGEN

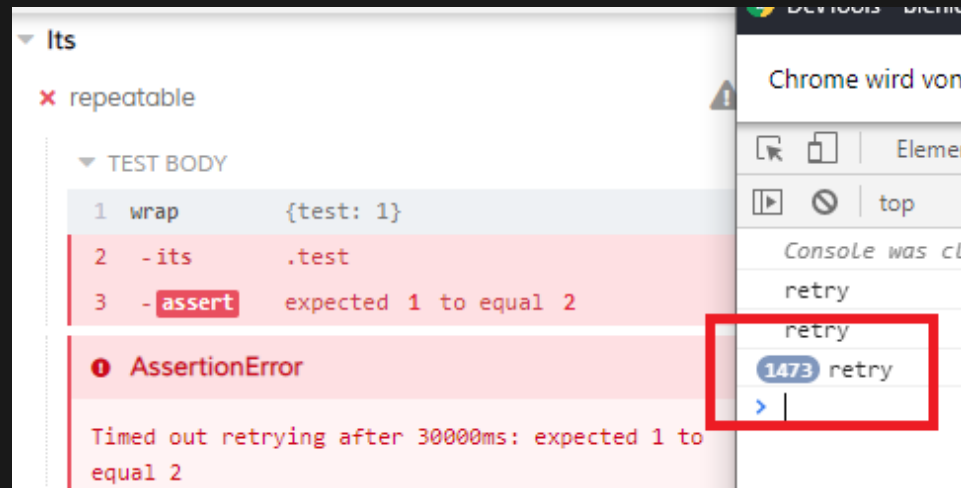
- ein reload() am Anfang schafft Abhilfe
- Cypress hat das ebenfalls auf dem Schirm
- Ist es überhaupt ein Problem?

# ITS

- `cy.its("...")` ruft das Feld mit dem entsprechenden Namen ab
- `cy.its()` ist auch repeatable

```
describe("Its", () => {  
  it("repeatable", () => {  
    var obj = {  
      get test() {  
        console.log("retry");  
        return 1;  
      }  
    }  
    cy.wrap(obj).its("test").should("eq", 2);  
  })  
})
```

# ITS



# CYPRESS CONTEXT

- Vergleich der zwei Hashes bei Login

```
1 Cypress.Commands.add("newTicket", () => {
2   cy.location().its("hash").as("loc");
3   cy.get("#shellnewticket").click().get("body").click();
4   cy.location().should(function(loc2) {
5
6     expect(this.loc).not.eq(loc2.hash);
7   }).then(loc2 => {
8     const identifier = loc2.hash.match(/.*(n[0-9]+)/)[1];
9     return identifier;
10  });
11 });
```

- spart eine Einrückungsebene
- erfordert aber function Syntax

# BEDINGTE PRÜFUNG

- wenn xyz dann mach dies oder jenes
- Achtung: Nicht unbedingt Best Practices, aber in Praxis notwendig
- hilfreich: `.then($e => ....) => JQuery!`

# ÜBUNG

- Checkbox anhacken
- möglichen vorherigen Status beachten
- ggf neue Person anlegen

```
2 describe("should", () => {
3     it("test", () => {
4         cy.login("developer", "Test12")
5         cy.visit("http://biehler-josef.de:9500/HD/#/person/
6         cy.get("div").contains("Neues System Feld 1").paren
7         .then($e => {
8
9             debugger;
10            if ($e.find("img[src*='CheckBox_No']").length >
11                cy.wrap($e).click();
12            } else if ($e.find("img[src*='CheckBox_Yes']").
13                // do nothing
14            }
15        })
16    })
17 })
```



# BEDINGTE PRÜFUNG - LÖSUNG

- Cypress baut nur auf JQuery auf
- innerhalb von then/should kann ALLES getan werden, was auch mit JQuery geht

```
1 describe("should", () => {
2     it("test", () => {
3
4         cy.login("developer", "Test12");
5         cy.visit("http://biehler-josef.de:9500/HD/#/person/");
6         cy.get("div").contains("Vielleicht").parent()
7             .find("input-check-box-component").wait(500)
8             .then($e => {
9                 if ($e.find("img[src*='CheckBox_No']").length > 0)
10                     cy.wrap($e).click();
11             } else if
12                 ($e.find("img[src*='CheckBox_Yes']").length > 0)
13                     // do nothing
14             }
15     })
16 })
```

# "MEINE LETZTEN TICKETS" PRÜFEN

- passt die Datumsanzeige "vor etwa x Stunden"
- wie umsetzen?
- Daten erzeugen -> zu jung
- drauf verlassen, dass da schon Tickets da sind -> zu instabil
- Datumswerte in DB faken -> zu viel verknüpfte Daten

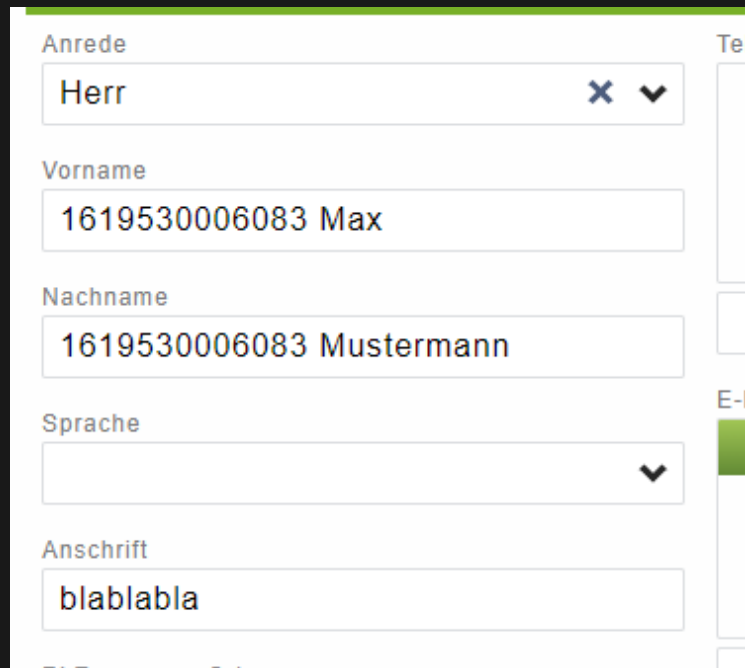
# LÖSUNG: REQUESTS MOCKEN

- `cy.intercept()`

# REQUESTERGEBNIS BEEINFLUSSEN

```
1
2 it("test", () => {
3     cy.login("developer", "Test12");
4     cy.visit("http://biehler-josef.de:9500/HD");
5     cy.intercept("GET", /*.*GetPerson.*/i, {"address":{"city"
6     cy.visit("http://biehler-josef.de:9500/HD/#/person/5");
7     cy.wait(50000);
8 })
```

# REQUESTERGEBNIS BEEINFLUSSEN



A screenshot of a web form with a light gray background and a green header bar. The form contains several input fields and a dropdown menu. The fields are labeled 'Anrede', 'Vorname', 'Nachname', 'Sprache', and 'Anschrift'. The 'Anrede' field has a dropdown menu with 'Herr' selected. The 'Vorname' field contains '1619530006083 Max'. The 'Nachname' field contains '1619530006083 Mustermann'. The 'Sprache' field is a dropdown menu with a downward arrow. The 'Anschrift' field contains 'blablabla'. To the right of the form, there are labels for 'Tel' and 'E-M'.

Anrede  
Herr

Vorname  
1619530006083 Max

Nachname  
1619530006083 Mustermann

Sprache

Anschrift  
blablabla

Tel

E-M

# GROSSE JSON OBJEKTE

- was, wenn ich nur einen Wert ändern möchte, ohne dafür das ganze Objekt anzugeben?
- Request senden und response bearbeiten

```
cy.login("developer", "Test12");  
cy.visit("http://biehler-josef.de:9500/HD");  
cy.intercept("GET", /.*GetPerson.*/i, r => {  
  r.reply(interceptor => {  
    interceptor.body.address.city = "Arsch der Welt";  
  })  
})
```

# AUF REQUEST WARTEN

- Sinnvoll!

```
1
2 cy.login("developer", "Test12");
3 cy.visit("http://biehler-josef.de:9500/HD");
4
5 cy.intercept("GET", /.*GetPerson.*/i, r => {
6     r.reply(interceptor => {
7         interceptor.body.address.city = "Arsch der Welt";
8     })
9 }).as("getperson");
10 cy.visit("http://biehler-josef.de:9500/HD/#/person/5");
11 cy.wait("@getperson").then(r => {
12     assert.equal(r.response.body.address.city, "Arsch der W
```

# TICKETPOOL MIT UNTERSCHIEDLICHEN FILTERN TESTEN

- Daten mocken abhängig vom Requestbody

```
1
2  const response = {
3      moreItemsAvailable: false,
4      totalRowCount: 1
5  }
6  const tOpen = {"id":"676","type":"Ticket","selected":false,
7  const tInprocessing = {
8      type: "Ticket",
9      selected: false,
10     id: "2",
11     data: {
12         ...tOpen.data,
13         aggregatedColumn_Ticketinfo: {
14             ...tOpen.data.aggregatedColumn_Ticketinfo,
15             id: 2,
16             title: "ticket in bearbeitung",
```



# AUF X. REQUEST WARTEN

- z.b. zweiten loadData Request
- möglich durch numerisches Suffix im alias

```
cy.intercept("POST", /*.*loadData.*/*).as("loadData");
cy.login("developer", "Test12").then(() => {
  cy.visit("http://biehler-josef.de:9500/HD/#/pool/")
  cy.get("#generic-pool-result-view .refresh").click()
  cy.wait("@loadData.2");
})
```

**DEMO**

# FIXTURES

- Daten auf diese Weise zu mocken kann nervig sein
- evtl besser: Fixtures
- Standardverzeichnis: "cypress/fixtures"

```
1
2  [
3    {
4      "lastActionDate": "2020-11-13T19:13:28.748Z",
5      "ticket": {
6        "id": "1234-ticketid",
7        "title": "test ticket",
8        "category": "test kategorie"
9      }
10   }
11  ]
```

# FIXTURES

```
2 describe("Last edited tickets", () => {
3     beforeEach(() => {
4         const d = new Date();
5         d.setHours(d.getHours()-15);
6
7         cy.intercept("GET", /.*GetLastEditedTickets.*/,
8             { fixture: "test.json" })
9         cy.login("developer", "Test12");
10    });
11
12    it("should show date string", () => {
13        cy.get(".last-edited-tickets-list")
14            .should("contain", "1234-ticketid");
15    });
16 })
```

# FIXTURES - CY.FIXTURE()

- Daten können ebenfalls über `cy.fixture()` geladen werden

```
it("test2", () => {  
  cy.fixture("test.json").then(f => {  
    debugger;  
  })  
})
```












# PROBLEM: TESTEN, DASS REQUEST NICHT KOMMT

- Szenario: Feldvorschläge
- Keine Vorschläge bei bestehendem Ticket
- Idee: try/catch

Titel 4/255

Test

Beschreibung

**B** *I* U ~~S~~           

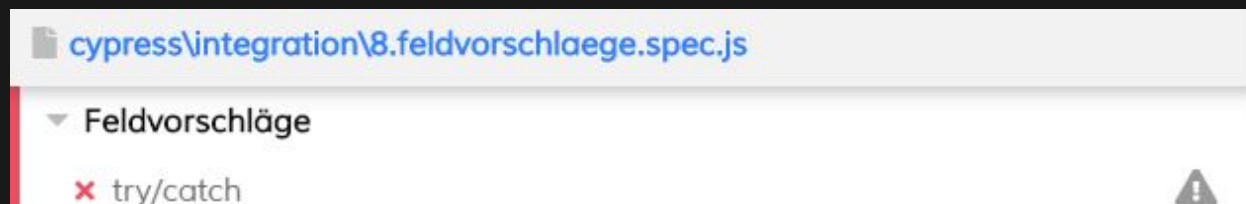
Dieselgenerator |

Manu's QA-Kategorie1 > Manu 1



# PROBLEM: TESTEN, DASS REQUEST NICHT KOMMT

- Richtiger Weg: `cy.on("error")`
- mit `cy.on()` auf Events lauschen
- <https://docs.cypress.io/api/events/catalog-of-events.html>





```
it("cy.on", done => {
  cy.on("fail", error => {
    if (error.name === "CypressError"
        && error.message.toString()

        .match(/.*timed out waiting `.*` for the 1st request to the ro
        // calling done forces cypress to turn test to

        done());
  });

  cy.intercept("POST", /.*getFieldRecommendationsFromText.*/
  cy.login("developer", "Test12");
  cy.visit("http://biehler-josef.de:9500/HD/#/ticket/1").wai
```

# ABER ACHTUNG

- Aufruf von `done()` ermöglicht nicht das zurückkehren an den Ort der Exception

# ÜBUNG

- Auf Request *getTicket* warten
- *title* des Tickets auf *Spartakiade* abändern
- <http://biehler-josef.de:9500/HD/#/ticket/1> aufrufen
- Tickettitel abprüfen

```
describe("test", () => {
  it("test", () => {
    cy.login("developer", "Test12");
    cy.intercept("GET", /.*getTicket.*/i, req => {
      req.continue(interceptor => {
        interceptor.body.title = "Spartakiade";
      })
    })
    cy.visit("/#/ticket/1")
      .get("#Title1 input").should("have.value", "Sparta")
  })
})
```

# REQUESTS MOCKEN - FAZIT

- auf Request warten ist essentiell in SPA
- Mittels Bearbeiten der Configuration kann der Client gezielt gesteuert werden, ohne viel Code mocken zu müssen
- Beispiel: Feldvorschläge

```
loginThroughAPI(null, ctx, {  
    modifyClientConfig: (c: ApplicationConfig): void => {
```

# HDA NAVIGATION: SIND ALLE LINKS KLIICKBAR?

- ablösen von manuellem Test: Durchklicken durch komplette Navigation
- jeden Link anklicken und Überschrift prüfen
- aber was bei Submenüs?

# HDA NAVIGATION: LÖSUNG

- Geschickte Kombination von Cypress & JQuery

```
1
2 let header = "dummy value";
3
4 cy.get("#mainSideBar li:has(a):not(:has(.fa-external-link))
5
6     const link = cy.wrap($e);
7     link.click();
8     const $subMenu = $e.find("li");
9
10    if ($subMenu.length > 0) {
11        cy.wrap($subMenu).should("be.visible");
12    } else {
13        cy.get(".container-fluid.page-heading").should($hea
14            const headerText = $header.text();
15            expect(headerText).not.equal(header);
```

# TICKETWEITERLEITUNG TESTEN

- Ticket in Team1
- Weiterleitung zu Team2
- IsNewInTeam wird auf true gesetzt nach einer gewissen Zeit
- Asynchron!
- Test wird flaky



# ASYNCHRONITÄT IN APP TESTEN

## - ERSTER VERSUCH

```
3 it("test", () => {
4     let ticket;

5     cy.login("developer", "Test12")
6         .then(async () => ticket = await createTicket("test
7         .then(() => cy.visit("/#/ticket/" + ticket.id))
8         .then(() => {
9             cy.get("#actionButton_views\\.Ticket\\.\\.ActionBu
10             cy.get("[id*=stopProcessing_changeTeamTicketAc
11             openSelect2("[data-ui-locator*=teamwidget] div
12             cy.get("[id*=changeTeamInvolvingAction]").choi
13             cy.get("[id*=changeTeamInvolvingAction] .label
14             cy.get("#actionButton_views\\.Ticket\\.\\.ActionBu
15         })
16         .then(() => ticketPool(ticket.title))
```

# ASYNCHRONITÄT TESTEN

- Möglichkeit 1: Wait()
- Allerdings Schlecht -> Testlaufzeit
- Möglichkeit 2: Warten bis Aktion verarbeitet ist

# ASYNCHRONITÄT TESTEN - APP ANPASSEN

- App mit spezifischem Code anreichern, um Testen zu erleichtern

```
export function waitUntilActionIsProcessed(systemAction, ticketId) {  
  function check() {  
    return get("http://biehler-josef.de:9500/api/  
      AutomaticTestSupport/IsActionProcessed",  
      { systemAction: systemAction,  
        ticketId: ticketId });  
  }  
  
  return repeat(400, 250, () => check(this));  
}
```

# METADATEN ERZEUGEN?

- Testen, ob InvolvingAction auftaucht bei Teamwechsel

# METADATEN ERZEUGEN?

```
1
2 function provideMetadata(data, identifier) {
3   return cy.token().then(t => {
4     return cy.request({
5       method: "POST",
6       url: `http://biehler-josef.de:9500/api/Automati
7       headers: {
8         "Authorization": "Bearer " + t,
9         "X-Auth-ClientId": "local-test-hd"
10      },
11      body: data
12    })
13  });
14 }
15
```

# MIT SPA FRAMEWORKS

## INTERAGIEREN

- drei Fälle: Events werfen & Queue & Events abhören

# EVENTS WERFEN

- Test: Betroffener CI-Kontakt entfernen, wenn CI gelöscht wird
- Event simulieren

# EVENTS WERFEN

Code in HD:

```
if (window.Cypress) {  
  // EventAggregator ist eine Klasse von uns  
  window.eventAggregator = aurelia.container.get(EventAggreg  
}
```



# EVENTS WERFEN

Test:

```
1
2 function publishAureliaEvent(event, data) {
3     cy.window().then(win => win.eventAggregator.publish(event, data))
4 }
5
6 // ticket anlegen mit betroffenen kontakt
7 it("test", () => {
8     cy.login("developer", "Test12");
9     cy.visit("/#/ticket/29")
10    cy.get(".calling-card").should("contain", "ci")
11    publishAureliaEvent(`entity:configurationitem:deleted:9`)
12    cy.get(".calling-card").should("not.contain", "ci")
13 })
```

# EVENTS WERFEN - VORTEILE

- keine Websockets notwendig
- Vermeiden von Fehlerquellen
- Deterministisches Verhalten

# FALL 2: QUEUE

- Aurelia legt Aktionen in Queue ab
- z.b. Rendern von for Schleife
- Wann fertig?

# FALL 2: QUEUE

## Änderungen an HD:

```
if ((window as any).Cypress) {  
    (window as any).aureliaTaskQueue = (window as any).taskQueue;  
    DependencyResolver.resolve<TaskQueue>(TaskQueue);  
}
```

## Funktion in Cypress:

```
function waitForTaskQueue() {  
    const randomFieldName = new Date().getTime().toString();  
    cy.window().then(win => win.aureliaTaskQueue.queueTask(()  
        win[randomFieldName] = true));  
    cy.window().should(win =>  
        expect(win[randomFieldName]).to.equal(true));  
}
```

# EVENTS ABHÖREN

- Bei bestimmten Sachen werden Events geworfen am Client
- Beispiel: Tabwechsel
- Könnte man das nutzen zur Testunterstützung?

# EVENTS ABHÖREN - BEISPIEL

- Wechsel zu neuem Tickettab
- Prüfen, ob korrekter Tab angezeigt wird



# EVENTS ABHÖREN

Wann kann dies hilfreich sein?

- testen, dass etwas nicht passiert zu gegebenen Zeitpunkt
- doppelter Timeout (Event + darauf folgender Command!)
- Sicherstellen, dass passiert, was man erwartet
- Wenn Events getestet werden sollen



# MIT OS INTERAGIEREN

- z.b.: Existiert Datei? imap-simple Lib einbinden,...
- in Testcase NICHT möglich (logisch, wird im Browser ausgeführt)

# MIT OS INTERAGIEREN

- Lösung: `cy.task()`
- Beispiel: `"doesFileExist"`
- NodeJS Thread

# PLUGIN.JS

```
const fs = require("fs");

module.exports = (on, config) => {
  on("task", {
    doesFileExist({ path }) {
      return fs.existsSync(path);
    }
  })
}
```

# TEST

```
describe("task", () => {  
  it("test", () => {  
    cy.task("doesFileExist", { path: "C:\\Users\\jbiehler\\"  
  })  
})  
})
```

# ÜBUNG

- Task erstellen um cwd auszulesen

# LÖSUNG

```
// plugins
module.exports = (on, config) => {
  on("task", {
    cwd() {
      return process.cwd()
    }
  })
}

// spec
it("test", () => {
  cy.task("cwd").then(cwd => {
    Cypress.log({
      displayName: "CWD",
      message: "cwd ist " + cwd,
    })
  })
})
```

# ACHTUNG VOR INSTANCEOF

- Problem: HDC Ticketliste testen
- Daten (Liste an Tickets) mocken





cypress/integration/30.array.spec.js

test

✓ test

TEST BODY

📌 window

2 - assert expected true to be true

✗ test 2



TEST BODY

1 assert expected false to be true

❗ AssertionError

expected false to be true

# CHROME DEBUGGER PROTOCOL

- Beispiel: "Drucken" Ansicht prüfen
- kein Fall für HD, aber privater Natur

# CHROME DEBUGGER PROTOCOL

- *chrome.exe --remote-debugging-port=9222 --user-data-dir=....*
- *C:\Program Files (x86)\Google\Chrome\Application*
- *C:\Program Files (x86)\Microsoft\Edge\Application*
- Alle debug Ziele: <http://localhost:9222/json>
- WS URL kopieren
- Test mit:

## Testbefehl:

```
{  
  "id": 1,  
  "method": "Page.navigate",  
  "params": {  
    "url": "https://de.wikipedia.org/wiki/Softwareentwickl  
  }  
}
```

# PRINT MEDIAQUERY AKTIVIEREN

```
{  
  "id": 1,  
  "method": "Emulation.setEmulatedMedia",  
  "params": { "media": "print" }  
}
```

# CDP + CYPRESS LIMITS

- Nur Chrome unterstützt CDP vollständig
- Cypress könnte ebenfalls CDP Session starten (Port Konflikt)
- Chrome State resetten

# CDP + CYPRESS

Im Plugins file:

- chrome-remote-interface einbinden (macht vieles einfacher)
- before:browser:launch event behandeln, um Port zu erhalten & Startargumente zu manipulieren
- Tasks für CDP definieren





# WEITERE ANWENDUNGSGEBIETE VON CDP

- Native Events absetzen
- Hover (CSS) triggern
- ....

# MEHRERE KUNDEN ABTESTEN

- wie können mit einem Cypress Projekt unterschiedliche Kundenanforderungen getestet werden?
- z.b. Ticket schließen mit/ohne Gründe
- versandlabel drucken durch Dienstleister

# BEIM TEST ATTRIBUTE ANGEBEN

- wie können Tests explizit Verhalten in HD steuern?
- Classifier ausschalten
- ES ausschalten
- Websockets deaktivieren
- .....

# TESTS NUR AUF BESTIMMTEN ENVIRONMENTS

- wie können Tests nur auf Release ausgeführt werden, nicht aber auf den anderen Systemen
- während andere Tests überall ausgeführt werden soll

# LÖSUNG FÜR ALL DAS

- Steuerung der Tests mittels Tags & Umgebungsvariablen
- WICHTIG: ggf Filterung der Testergebnisse!
- siehe Code 25.brand-tags.spec.js

# STEUERN DES VERHALTEN VON HD

- Authentifizierung
- Wechsel der Auth Tokens je nach benötigten Nutzer
- Erweiterung des Tagkonzepts

# VISUELLE TESTS

- Tacho testen
- <https://docs.cypress.io/guides/tooling/visual-testing#Functional-vs-visual-testing>
- Empfehlung: <https://github.com/mjhea0/cypress-visual-regression>

# STUBS & SPIES

- auch möglich
- ersetzen von Date, ...
- hatten bisher aber keinen Verwendungszweck



# TESTS RETRY

- hilft gegen Flaky Tests
- Einfacher Parameter in npm script
- Achtung: "State zwischen Tests" Problem!

# DASHBOARD



## Supercharged Velocity

Use the power of parallelization to run more tests, faster

- ✓ Decrease test maintenance by [up to 75%](#)
- ✓ Identify and resolve issues in CI more quickly
- ✓ Ship and release more features, with greater confidence



## Reduced Execution Time

Accelerate test speed, and spend more time developing

- ✓ Run tests in [multiple browsers](#) by grouping test runs for Firefox, Chrome, Edge, and more
- ✓ Utilize the power of Cypress [parallelization](#) within your groups to execute tests [up to 5x faster](#)
- ✓ Group tests by environment, package, custom configuration, and more to easily organize results



## Increased Confidence

Unlock critical data and results for your team

- ✓ See the exact point of failure in your UI with auto-generated screenshots of test failures
- ✓ Visualize the impact of every error by watching full video recordings of any test you run
- ✓ Get notified instantly when tests fail with our GitHub and Slack Integrations

# CYPRESS STUDIO

- Interaktion auf Seite als Test speichern
- geht derzeit nicht mit Firstanswer HD
- <https://docs.cypress.io/guides/core-concepts/cypress-studio>
- siehe Demo

# BEFORE/AFTER RUN/SPEC EVENTS

- <https://docs.cypress.io/api/plugins/before-run-api>
- <https://docs.cypress.io/api/plugins/after-run-api>
- <https://docs.cypress.io/api/plugins/before-spec-api>
- <https://docs.cypress.io/api/plugins/after-spec-api>

# AUTOMATISIERUNGEN

- Cypress kann noch mehr
- Crawl von Slack
- <https://github.com/gabbersepp/slack-crawler>