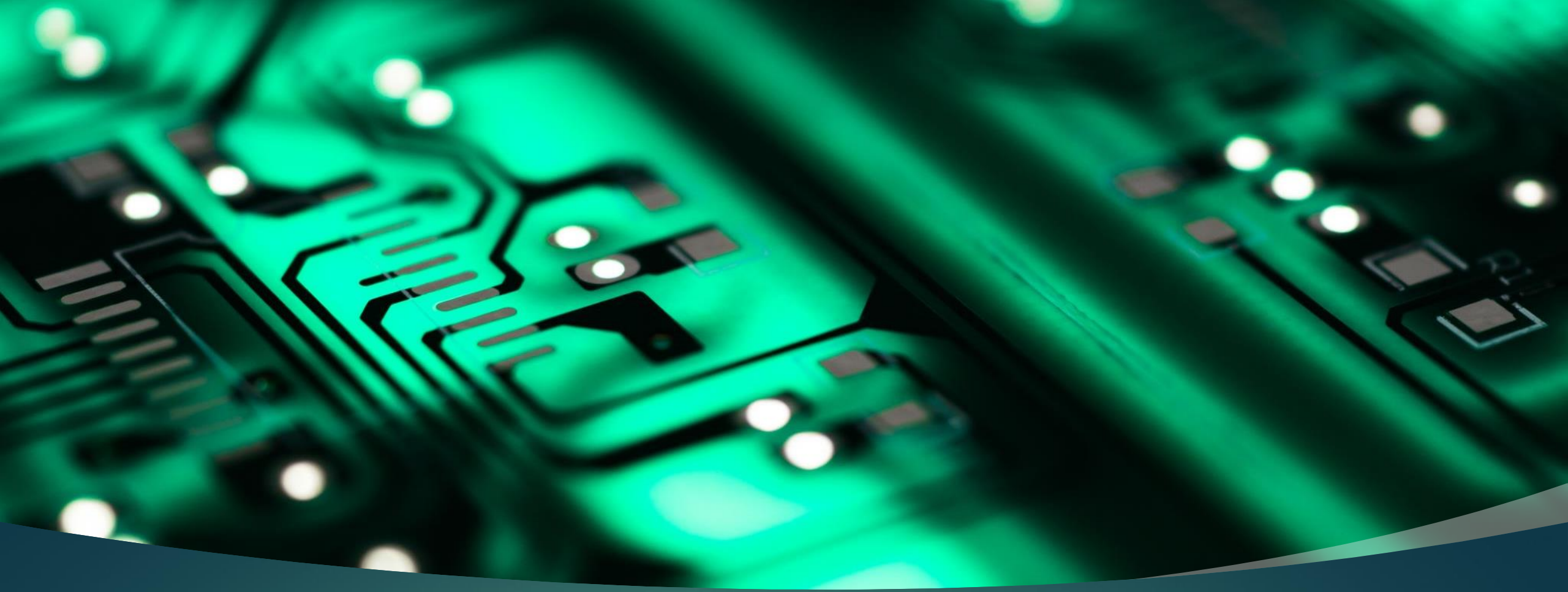
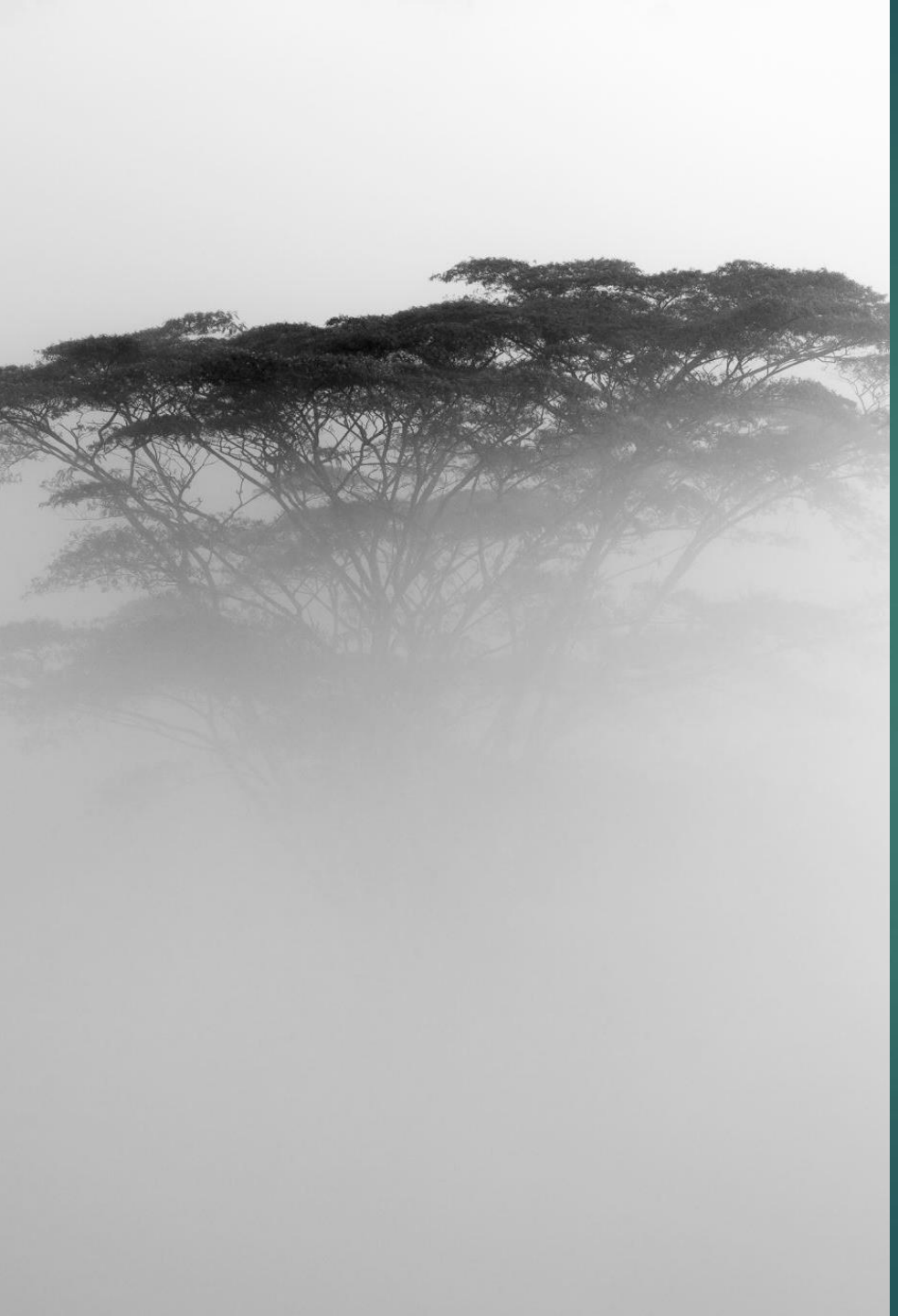


Testowanie Automatyczne II

WYKŁAD 2



Konfigurowanie plikami

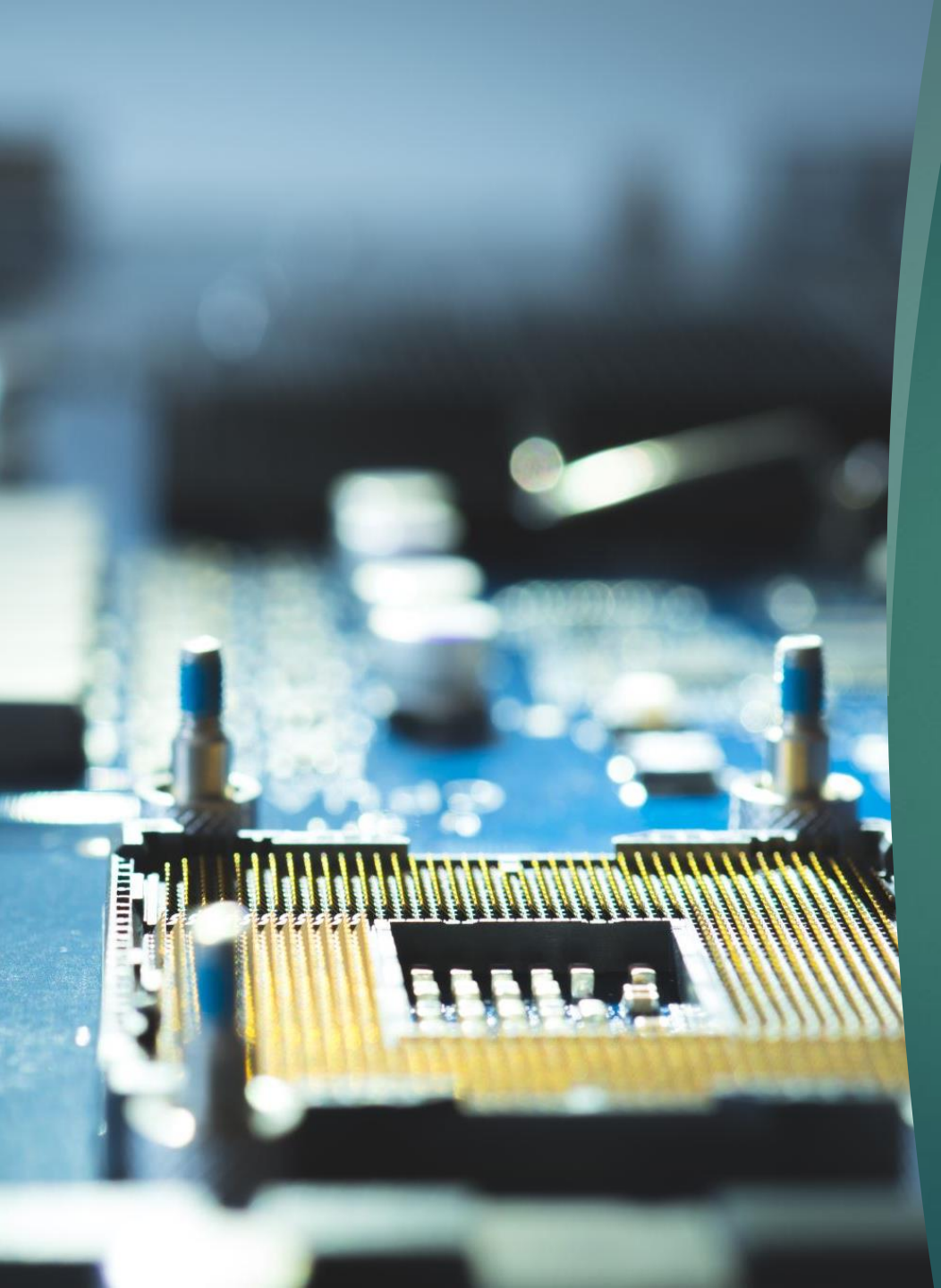
- Cypress od wersji 10+
  - TypeScript: cypress.config.ts
  - JavaScript: cypress.config.js
- Pliki cypress.config
  - Głównym plikiem konfiguracyjnym
  - Może być wskazywany jak i częściowo nadpisywany
  - Przez zmiany w tym pliku zmieniamy domyślne zachowania

Cypress.json

- Kiedyś plik na start był „pusty”
  - Zawiera tylko {}
- Teraz zawiera podstawową strukturę
- Plik można wskazać przy uruchomieniu cypress
  - npx open --config-file {adress pliku}
  - npx run --config-file {adress pliku}

Konfiguracja

- Można również na poziomie wiersza poleceń nadpisać elementy konfiguracji
  - npx run --config {nazwa = wartość },
  - npx open --config {nazwa = wartość },
- Niektóre polecenia również pozwalają na dostarczenie własnych konfiguracji
  - cy.get('a',{timeout:10000})



Zestawy opcjii

- Co można definiować
  - Cypress dzieli to na 11 grup zainteresowań
  - Global
  - Timeouts
  - Folders / Files
  - Videos
  - Downloads
  - Viewport
  - Actionability
  - Node version
  - Experiments

Kilka opcji

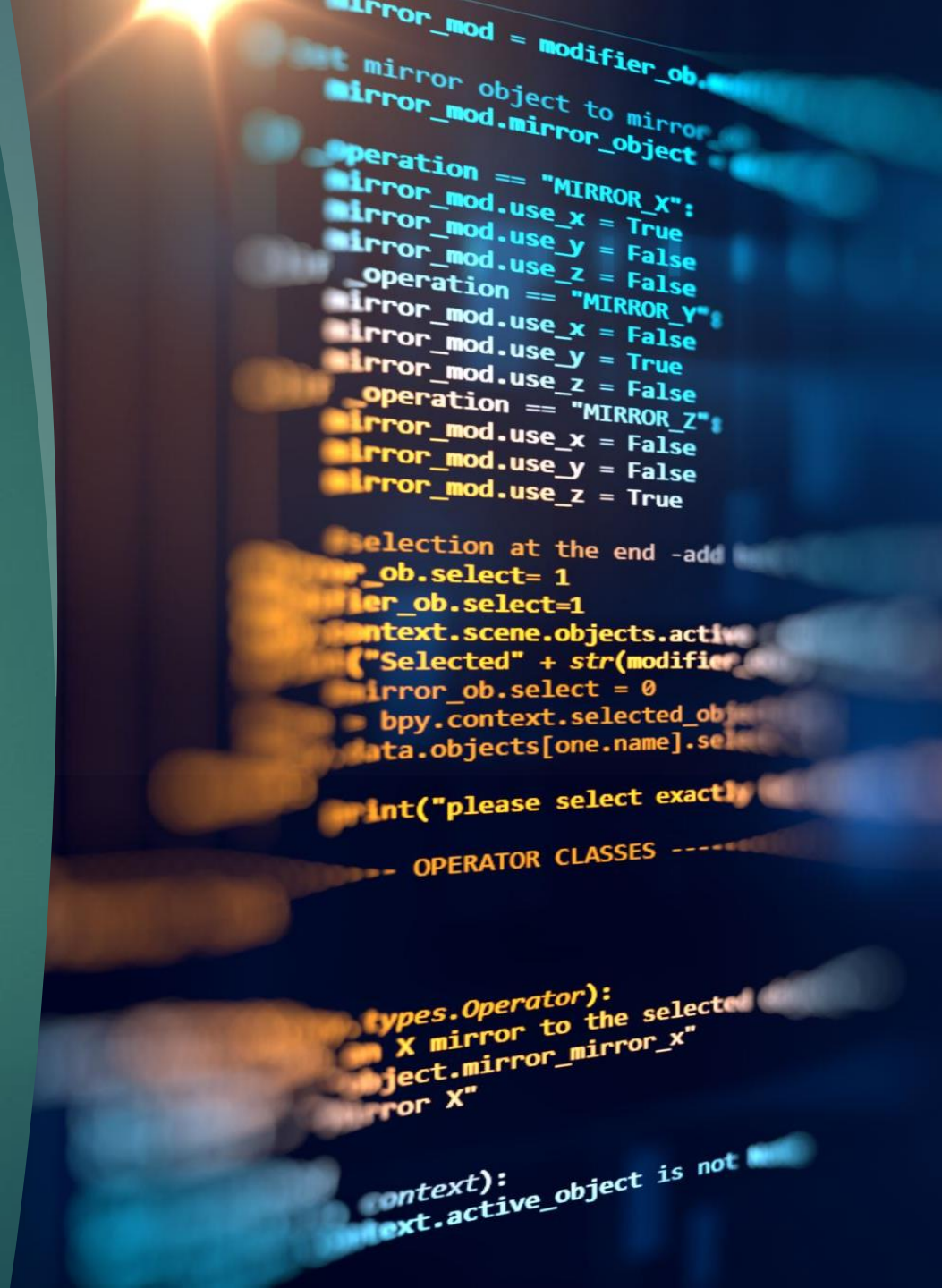
- baseUrl
  - Główny adres dla solucji
- Retries
  - Ile ponowień po tym jak test zwrócił negatywny wynik ma podjąć cypress
- watchForFileChanges
  - Czy cypress ma sprawdzać czy są zmiany w plikach
- defaultCommandTimeout
  - Domyślny timeout dla poleceń

Przykładowe Opcje Video/Screenshoot

- screenshotOnRunFailure
- video
  - Czy ogólnie nagrywać
- videoUploadOnPasses
  - Czy wysyłać nagrania jeżeli test przechodzi

Chrome Web Security

- chromeWebSecurity
  - Bardzo potężna opcja często potrzebna gdy:
    - Mamy cross domain origin wbudowany w stronę
    - Gdy wiemy że JS na stronię rzuca wyjątkami ☺
    - Gdy testujemy strony https bez certyfikatu



Dodatkowe Opcje

- viewportHeight, viewportWidth
- Environment
  - Zmienne środowiskowe do używania w testach
  - Mogą posiadać własny plik
- Experimental
  - Tak naprawdę funkcjonalności testowe

Przykład Config file

JS cypress.config-accept.js > ...

```
1   const { defineConfig } = require("cypress");
2
3   module.exports = defineConfig({
4     e2e: {
5       chromeWebSecurity: false,
6       retries: 2,
7       watchForFileChanges: true,
8       defaultCommandTimeout: 3000,
9       screenshotOnRunFailure: true,
10      video: true,
11      videoUploadOnPasses: false,
12      viewportHeight: 1920,
13      viewportWidth: 1080,
14      setupNodeEvents(on, config) {
15        // implement node event listeners here
16      },
17    },
18  });
19
```

Fixtures

- Dane zamrożone na poziomie wczytania
- Wczytywane przez:
  - polecenie cy.fixture
  - import jeżeli mówimy o plikach json

Po co Fixtures

- Przechowujemy w nich stałe wartości które są nam potrzebne w testach
  - np.:
    - Dane użytkowników
    - Wartości do uzupełnień input
    - Spodziewane wyniki wykonania

Jak używać

```
1   /// <reference types="cypress" />
2   import users from '../../fixtures/users.json'
3
4   describe("Fixtures",() => {
5       beforeEach(function (){
6           cy.visit("http://www.google.com");
7           cy.fixture('users.json').as('UserData');
8           cy.fixture('users.json').then((users2)=>{
9               this.users2 = users2;
10          });
11      })
12      it('Loading fixtures',function (){
13          cy.log(users[0].address.city);
14          cy.log(this.UserData[0].address.city);
15          cy.log(this.users2[0].address.city);
16      })
17
```

Jak używać 2

```
1  /// <reference types="cypress" />
2  import users from '../../fixtures/users.json'
3  import profile from '../../fixtures/profile.json'
4
5  describe("Fixtures",() => {
6      beforeEach(function (){ ···
12     })
13     it('Loading fixtures',function (){        ···
17     })
18
19     it('Salting fixture',function (){
20         const d = new Date();
21         cy.log(profile.name.replace("[salt]",d.getUTCDate().toString()+"_"+(d.getUTCMonth()+1).toString()+
22                                         "_"+d.getHours().toString()+d.getMinutes().toString()+d.getUTCSeconds().toString()));
23     })
24 })
25
```

Aliasy

- Stosowane jako odłożenie wyniku polecenia lub obiektu na później
- Korzystamy z nich przez:
  - Jako łańcuch po poleceniu .as("nazwa")
  - W poleceniu przez @nazwa
  - Jako obiekt przez this.nazwa
- Obiekt przechowywany jest identyczny do zwróconego przez polecenie
- Aliasy są czyszczone przed każdym wykonaniem testu

Wtedy zwane Then

- Polecenie pozwalające na pracę z wynikiem poprzedniego polecenia.
- Polecenie Then musi być rozwiązane przed rozpoczęciem następnego polecenia zakolejkowanego
- Polecenie Then potrafi zwracać cypress object
  - Jeżeli nie zawiera return to wynik ostatniego polecenia wewnątrz funkcji

Then Przykład

```
cy.get('@label').then(($label) => {
    //debugger;
    cy.log($label);
    cy.get('ytd-rich-grid-row:first-child ytd-rich-item-renderer:nth-child(1) #video-title-link').click().wait(3000);
    cy.get('.ytd-watch-flexy > :nth-child(1) > .title > .style-scope', { timeout: 30000 })
        .invoke('text').then(($title) => {
            //debugger;
            expect($title).to.equal($label);
        })
    cy.get('.ytd-watch-flexy > :nth-child(1) > .title > .style-scope', { timeout: 30000 })
        .invoke('text').should('contain', $label);
    cy.get('.ytd-watch-flexy > :nth-child(1) > .title > .style-scope', { timeout: 30000 })
        .should('contain', $label);
})
```

Invoke

- Wywoływanie funkcji na poprzedzającym elemencie w łańcuchu
- Używamy do wywoływania jQuary functions
- Jeżeli polecenie poprzedzające zwróciło element możemy
  - Pobrać przy jego użyciu atrybut(attr), text(text), wartość(val)

Custom Commands

- Cypress pozwala na tworzenie własnych poleceń jak i ich nadpisywanie.
- Tworzone polecenia mogą być poleceniami
  - Bazowymi
  - Łańcuchowym
  - Hybrydowym

CC zwroty i wejścia

- Custom Command zawsze zwraca obiekt cypress
  - Jeżeli nie mamy return zostanie przekazany wynik ostatniego polecenia wykonanego przez command
- Custom Command może mieć wiele argumentów wejściowych
  - Jeżeli jest to polecenie przyjmujące poprzednika to pierwszy argument to poprzednik
  - Jeżeli nadpisujemy polecenie istniejące to pierwszy argument to oryginalne polecenie a drugi to potencjalny element

Lokalizacja

- Commands możemy dodać niezależnie w dowolnym pliku testowym
- Możemy również dodać je jako dostępne przez użycie pliku e2e.js i wskazanie w nim odpowiedniego importu

Przykład polecenia

```
Cypress.Commands.add('nothing',()=>{
    cy.log('doing nothing');
})
Cypress.Commands.add('returnAelements',()=>{
    cy.get('a');
})
```

```
Cypress.Commands.overwrite('type',(originalFn,element,text,options)=>{
    const d = new Date();
    let saltedtext = text.replace("[salt]",d.getUTCDate().toString()+"_"+(d.getUTCMonth()+1).toString()+
        "_"+d.getHours().toString()+d.getMinutes().toString()+d.getUTCSeconds().toString());

    return originalFn(element,saltedtext,options);
})

Cypress.Commands.add('checkElementVisibility', { prevSubject: 'element'}, (subject, options) => {
    return cy.wrap(subject).should('be.visible');
})
```

Przykład importu Commands

cypress > support > JS e2e.js > ...

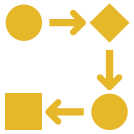
```
1   // ***********************************************************
2   // This example support/e2e.js is processed and
3   // loaded automatically before your test files.
4   //
5   // This is a great place to put global configuration and
6   // behavior that modifies Cypress.
7   //
8   // You can change the location of this file or turn off
9   // automatically serving support files with the
10  // 'supportFile' configuration option.
11  //
12  // You can read more here:
13  // https://on.cypress.io/configuration
14  // ***********************************************************
15
16  // Import commands.js using ES2015 syntax:
17  import './commands'
18
19  Cypress.on('uncaught:exception', (err, runnable)=>{
20      return false
21  })
22
23  // Alternatively you can use CommonJS syntax:
24  // require('./commands')
```

Ale po Co?

Zmniejszenie ilości powielanego kodu w wielu plikach testowych

Logowanie

Zamykanie powiadomienia cookie



Tworzenie poleceń do zadań specjalnych

Annominacja log by chronić hasła

Zasalanie danych by były unikalne

Poprawa czytelności plików testowych

Ułatwienie refaktoryzacji

Wzorce Projektowe

- Czym są wzorce projektowe
  - **Wzorzec projektowy** (ang. *design pattern*) – uniwersalne, sprawdzone w praktyce rozwiązanie często pojawiających się, powtarzalnych problemów projektowych. Pokazuje powiązania i zależności pomiędzy klasami oraz obiektami i ułatwia tworzenie, modyfikację oraz utrzymanie kodu źródłowego. Jest opisem rozwiązania, a nie jego implementacją.

Po co?

- Poprawa jakości
- Rozwiązywanie standardowych problemów
- Unikanie błędów

Page Object

- Wzorzec przyjęty jako standard porządkowania zestawów elementów do dalszego użycia w testach. Reprezentowany jako obiekt strony lub widoki i ich elementów od samego ciała testu.
- Ułatwia nam to
  - Naprawianie problemów z selektorami
  - Utrzymanie czytelności testu
  - Dalsze pracę nad rozwojem projektu testu
  - Refakturujące kodu

```
/// <reference types="cypress" />
...
class googlePage {

    _searchInputSelector = "input[name='q']";
    get searchInputSelector() {...
    }
    set searchInputSelector(value) {...
    }

    _personalizeButtonSelector = '#VnjCcb > .QS5gu';        You, 8
    get personalizeButtonSelector() {...
    }
    set personalizeButtonSelector(value) {...
    }

    _confirmCookiesSelector = '#L2AGLb > .QS5gu';
    get confirmCookiesSelector() {...
    }
    set confirmCookiesSelector(value) {...
    }

    _cookiesPopupSelector = '#CXQnmb';
    get cookiesPopupSelector() {...
    }
    set cookiesPopupSelector(value) {...
    }

    searchInput() {
        return cy.get(this.searchInputSelector);
    }

    personalizeButton() {
        return cy.get(this.personalizeButtonSelector);
    }

    confirmCookies() {
        return cy.get(this.confirmCookiesSelector);
```

Na czym Polega

- Tworzymy klasę
- Umieszczamy w niej
  - Funkcję które nam zwrócą obiekty z danego widoku
  - Możemy stworzyć zestaw selektorów w postaci własności które wykorzystamy w funkcjach

```
/// <reference types="cypress" />
import googlePage from '../Pages/googlePage.js';
import searchResultPage from '../Pages/searchResultsPage.

const google = new googlePage();
const searchResult = new searchResultPage();

Cypress.Commands.add('closeCookiesGoogle',()=>{…
})

Cypress.Commands.add('closeCookiesSearch',()=>{…
})

Cypress.Commands.add('closeCookies',($Page)=>{…
})

describe("PageObjects",() => {
    beforeEach(function (){…
    })
    it('Using PageObject',function (){
        //google.confirmCookies().click();
        cy.closeCookiesGoogle();
        google.searchInput().type('Wikipedia');
        google.searchInput().type('{enter}');
        searchResult.searchInput().clear().type('Wiki');
        searchResult.searchInput().clear().type('{enter}
    })
    it('Using PageObject',()=>{
        cy.closeCookies(google);
        //cy.nothing();
        //searchResult.confirmCookies().click();
    })
})
```

Na czym Polega

- Tworzymy klasę
  - Umieszczamy w niej
    - Funkcję które nam zwrócą obiekty z danego widoku
    - Możemy stworzyć zestaw selektorów w postaci proporcji które wykorzystamy w funkcjach
- W plikach testowych
  - Importujemy klasy
  - Tworzymy nowe obiekty przy użyciu klas
  - Odwołujemy się do elementów przez wywołanie odpowiednich elementów

Page object+

- Często specyficzne komendy dla strony przechowuje się w plikach podobnie nazwanych np.
  - GooglePage.js
  - GooglePageCommands.js

Wzorzec Memo

- Wzorzec Memo jest to pochodny wzorca singleton
- Umożliwia nam:
  - Jednorazową Inicjalizacje elementu/obiektu
  - Każde kolejne wywołanie spowoduje zwrot poprzednio stwożonegoobiektu

```
Cypress.Commands.add('memo',()=>{
    if (somevalue) {
        cy.log('I will return stored value')
        return cy.wrap(somevalue);
    }
    cy.log('I will return generated value')
    const random = Cypress._.random(10)
    cy.log(random);
    somevalue = { value: random}
    return cy.wrap(somevalue)
})

describe("Commands",() => {
    beforeEach(function (){
        cy.visit("http://www.google.com");
        cy.closeGooglePrefernces();
    })
    it('Using Command',function (){
        cy.nothing();
        cy.get('a').should('have.length',20)
        cy.get('body').checkElementVisibility();
        cy.returnAelements().should('have.length',20);
        cy.searchGoogle('Wikipedia[salt]');
        cy.memo().then(($value)=>{
            cy.log($value.value)
        })
        cy.memo().then(($value)=>{
            cy.log($value.value)
        })
        cy.lazy();
    })
})
```

Jak wygląda?

- Wpierw sprawdzamy czy obiekt istnieje
- Jeżeli istnieje zwaracamy obiekt
- Jeżeli nie istnieje to inicjalizujemy obiekt
  - Zapisujemy go w zmiennej
  - Zwracamy obiekt