

Testuj automatycznie swoją
stronę z Cypressem w mniej niż
1h

—
Sebastian Gągor

O mnie



Sebastian Gągor

W IT od 2017 roku

- * Programista PHP
- * Tester automatyczny
- * Scrum Master

<https://www.linkedin.com/in/sebastian-gagor/>



SEBASTIAN GĄGOR

Sebastian's Certifications

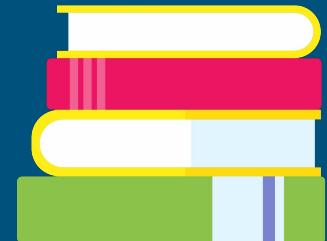


Classes Attended by Sebastian

Sebastian has no visible classes.

Plan prezentacji

1. Czym jest Cypress?
2. Cypress vs Selenium
3. Co jest nam potrzebne do uruchomienia Cypressa?
4. Instalacja Cypressa
5. Struktura Cypressa
6. Struktura testu - Mocha
7. Tworzymy pierwszy test - cypress runner
8. Hooki - before,after, beforeEach, afterEach skip, only
9. Sposoby wybierania elementów - przypominamy sobie cssa :)
10. Test formularza
11. Test nawigacji
12. Screenshoty
13. Dashboard Service - omówienie
14. Live coding
15. Q&A, rozszerzenie prezentacji o wnioski z prezentacji



Cypress



Czym jest Cypress?

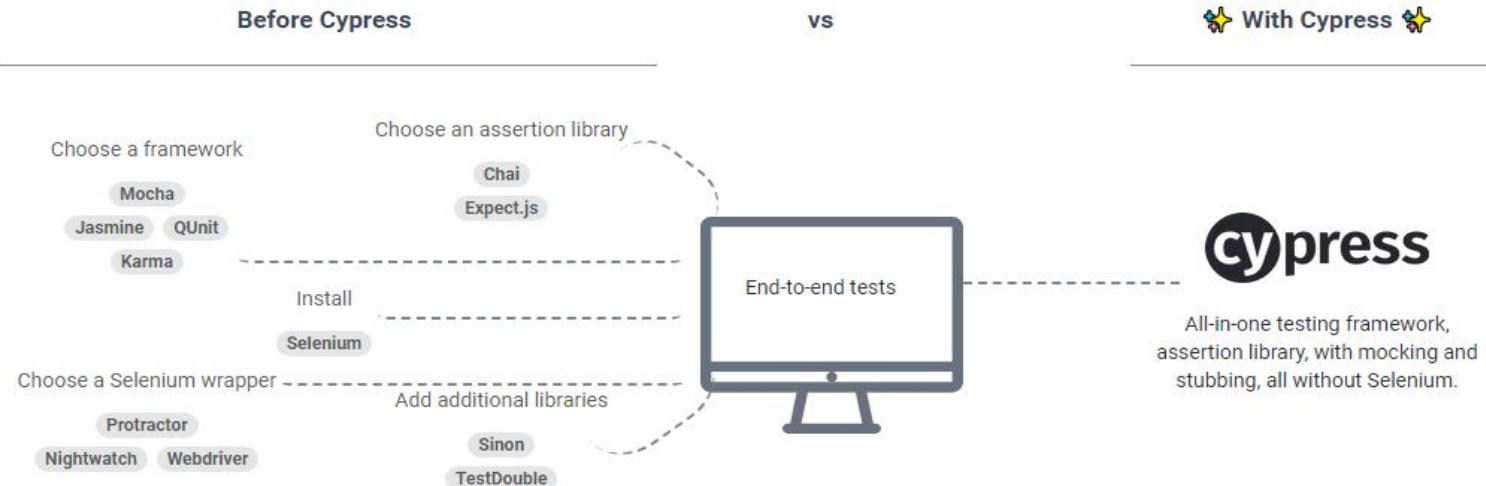
Cypress to narzędzie do testowania front-end nowej generacji stworzone dla nowoczesnych projektów webowych. Rozwiązuje kluczowe problemy, z którymi borykają się programiści i inżynierowie kontroli jakości podczas testowania nowoczesnych aplikacji.

Misja twórców

Naszą misją jest zbudowanie dobrze prosperującego ekosystemu open source, który zwiększa produktywność, sprawia, że testowanie jest przyjemnym doświadczeniem i generuje zadowolenie programistów. Czujemy się odpowiedzialni za wspieranie procesu testowania, który faktycznie działa.

Before Cypress vs with Cypress

Źródło: <https://www.cypress.io/how-it-works>



2. Cypress vs Selenium

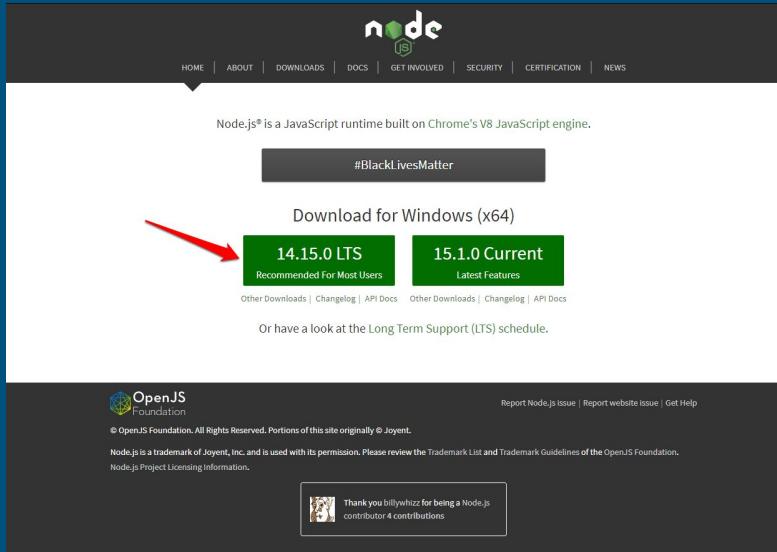
	Cypress	Selenium
Wspierane języki	Tylko JavaScript	Popularne języki jak Java, Python, Ruby, C#, PHP etc.
Wsparcie przeglądarek	Chrome, Edge, Firefox, Electron	Chrome, IE, Safari, Edge, Firefox
Wsparcie frameworków	Tylko Mocha JS	Wiele frameworków specyficzne dla języków (JUnit - Java, Cucumber - JavaScript etc)
Złożoność instalacji	Łatwa - niepotrzebne są żadne dodatkowe zależności ani pliki	Trudna - wymaga ściągnięcia driverów dla przeglądarek, skonfigurowania całego środowiska "od zera"
Dokumentacja & Społeczność	Intuicyjna dokumentacja, rosnąca społeczność	Gruntowna dokumentacja, społeczność z całego świata

Ograniczenia Cypressa

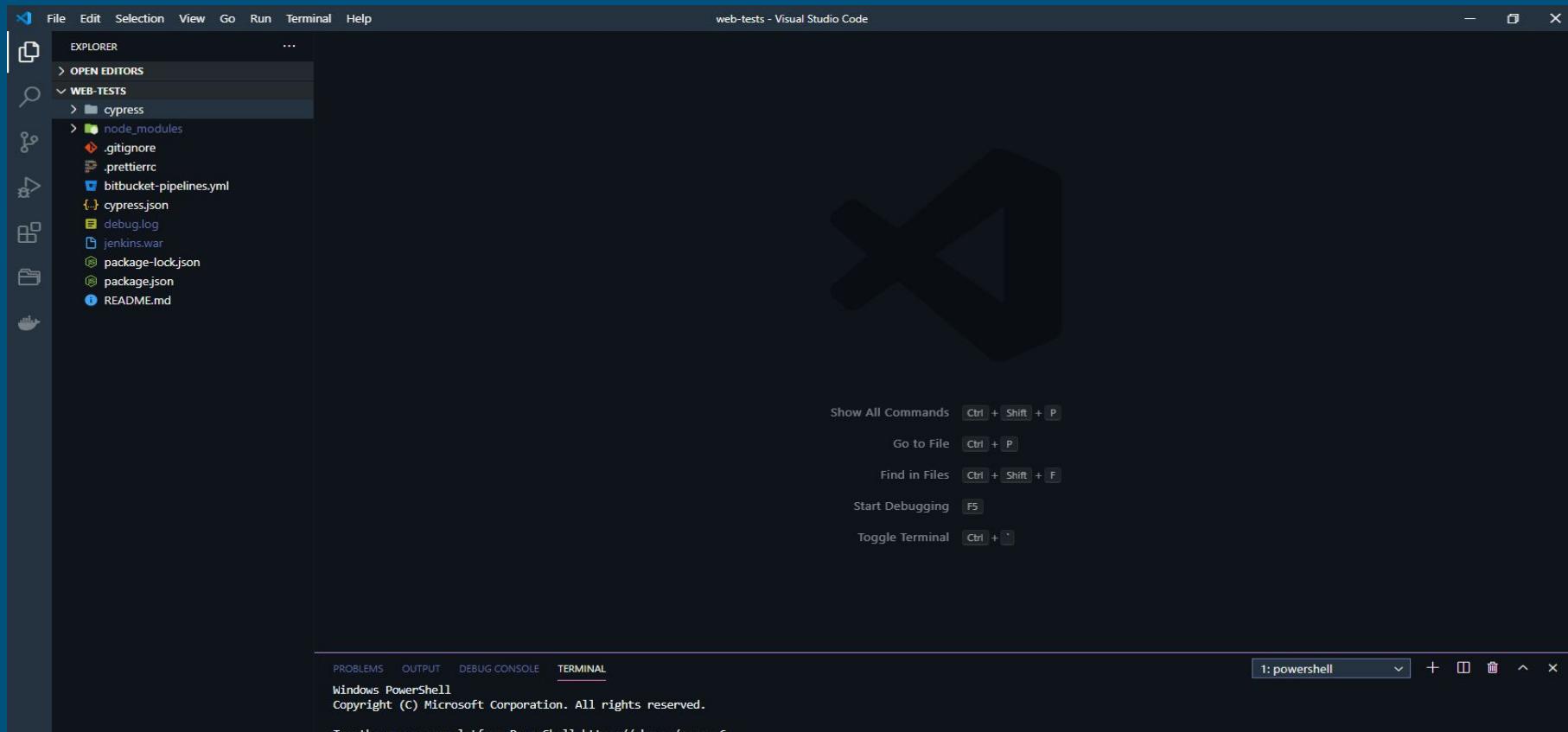
1. Nie można używać Cypressa do obsługi dwóch przeglądarek jednocześnie
2. Cypress nie obsługuje wielu kart
3. Cypress obsługuje tylko JavaScript do tworzenia testów
4. Brak wsparcia przeglądarek takich jak Safari czy IE.
5. Ograniczona obsługa iFrame

3. Co jest nam potrzebne do uruchomienia Cypressa?

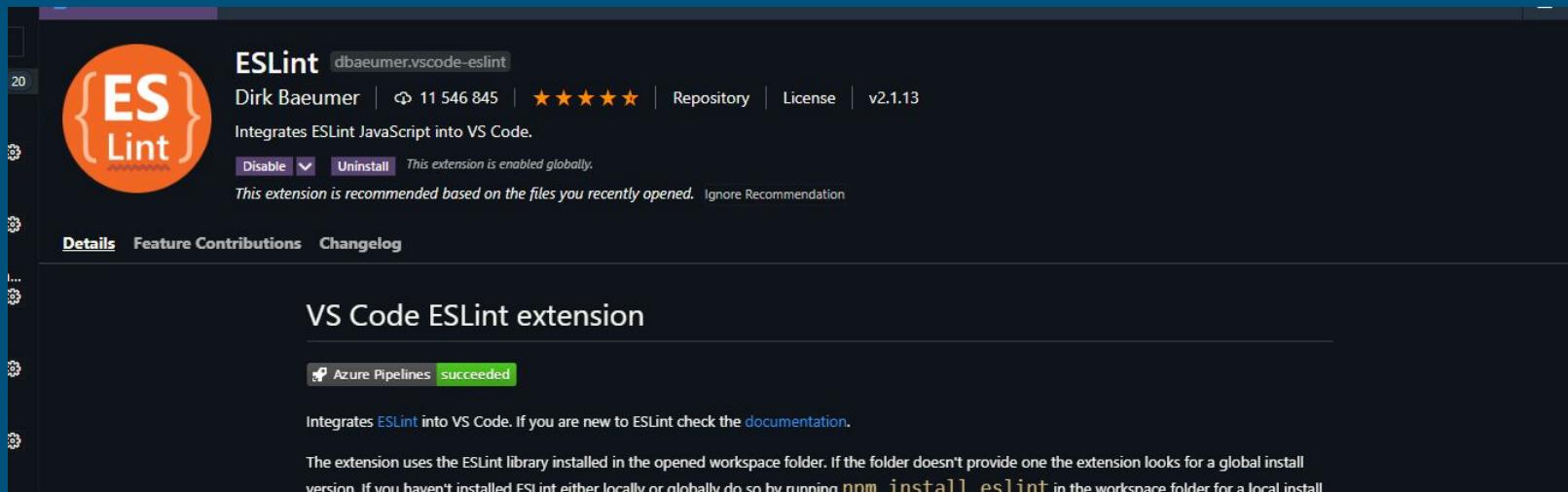
Aby zacząć korzystać Cypressa musimy pobrać i zainstalować Node.js
<https://nodejs.org/en/>



Z Cypressem dobrze współpracuje edytor kodu Visual Studio Code



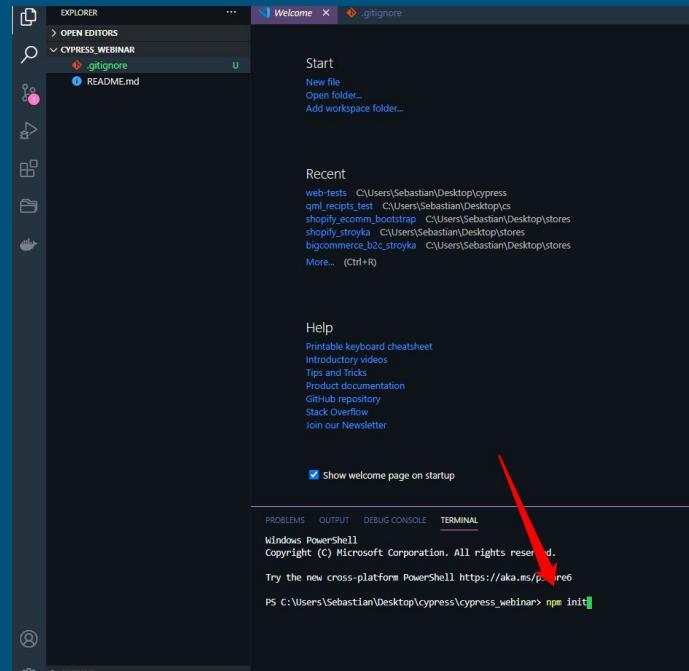
Przyda również się plugin do Visual Studio Code - ESLint



4. Instalacja Cypressa

W stworzonym pustym katalogu wpisujemy komendę:
npm init

Aby utworzyć plik package.json



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays a folder named 'CYPRESS_WEBINAR' containing a '.gitignore' file and a 'README.md' file. The Welcome screen on the right shows recent projects like 'web-tests', 'qml_recips_test', 'shopify_ecomm_bootstrap', 'shopify_stroyka', 'bigcommerce_b2c_stroyka', and 'bigcommerce_b2c_stroyka'. The bottom right corner of the interface has a red arrow pointing towards the terminal tab.

VS Code interface showing the terminal tab with the command 'npm init' entered:

```
PS C:\Users\Sebastian\Desktop\cypress_webinar> npm init
```

Aby zainstalować Cypressa i opcjonalnie plugin prettier wpisujemy komendę:

npm install cypress prettier

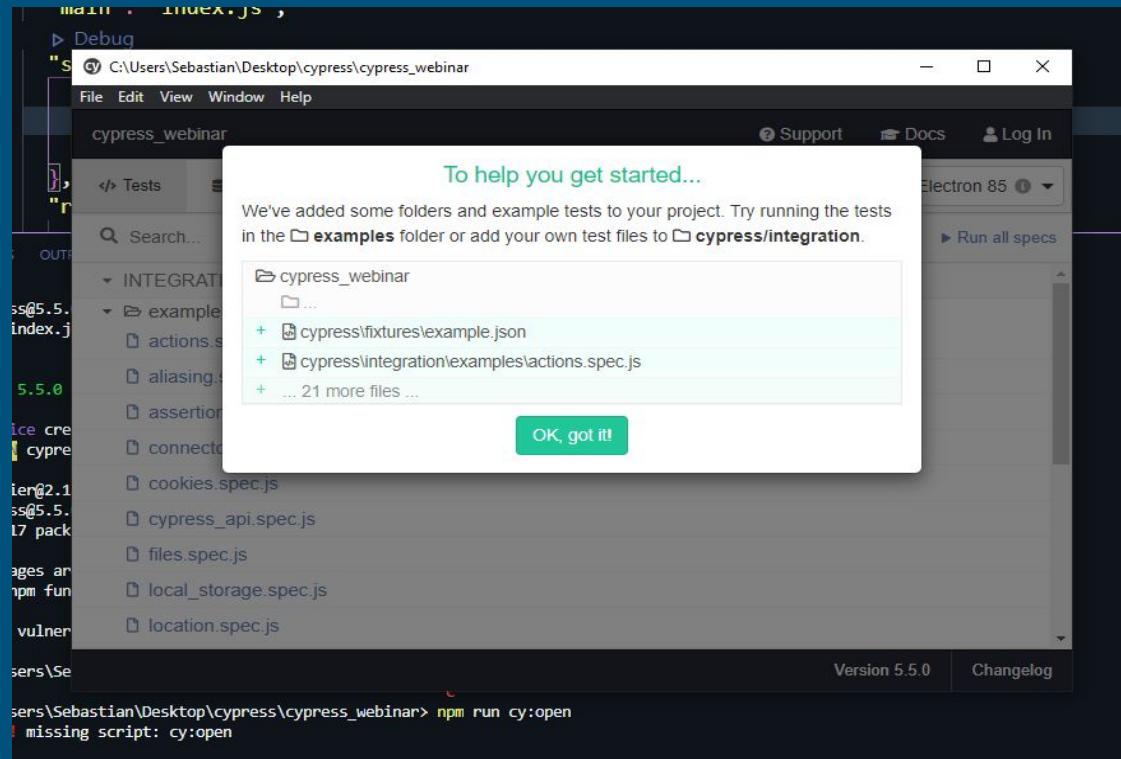
```
Is this OK? (yes)
PS C:\Users\Sebastian\Desktop\cypress\cypress_webinar> npm install cypress prettier ←
[.....] \ fetchMetadata: sill resolveWithNewModule cypress@5.5.0 checking installable status
```

Następnie dodajemy komendy **cy:open** oraz **cy:run** do package.json

```
package.json  ✓ scripts
{
  "name": "cypress_webinar",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "cy:open": "cypress open",
    "cy:run": "cypress run"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/sebb94/cypress_webinar.git"
  },
  "author": "Sebastian Gagor".
}
```

Cypress jest gotowy do uruchomienia - wpisujemy w terminalu **npm run cy:open**

Cypress pomyślnie uruchomił się po raz pierwszy oraz stworzył dla nas przykładowe pliki z testami w folderze integration/examples - polecam przeanalizować kod



Do ukończenia konfiguracji Cypressa warto w folderze z projektem utworzyć plik **.prettierrc**, a także podstawowa konfiguracja w pliku **cypress.json**. W folderze z Cypressem plik **tsconfig.json** - pliki będą dostępne na moim githubie - > https://github.com/sebb94/cypress_webinar

The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar: Shows the project structure under **CYPRESS_WEBINAR**, including **cypress** (with **fixtures**, **integration**, **plugins**, **support**, **tsconfig.json**), **node_modules**, **.gitignore**, **.prettierrc**, **cypress.json** (selected), **package-lock.json**, **package.json**, and **README.md**.
- OPEN EDITORS** tab bar: Shows files: package.json, .prettierrc, tsconfig.json, cypress.json (highlighted in purple), 1.spec.js, and .gitignore.
- cypress.json** content (shown in the main editor):

```
1 {  
2     "chromeWebSecurity": false,  
3     "waitingForAnimations": true,  
4     "viewportWidth": 1650,  
5     "viewportHeight": 1080,  
6     "waitForAnimations": true,  
7     "defaultCommandTimeout": 5000,  
8     "execTimeout": 60000,  
9     "pageLoadTimeout": 30000,  
10    "requestTimeout": 60000,  
11    "responseTimeout": 60000,  
12    "video": false,  
13    "failOnstatusCode": false  
14 }  
15
```

Red arrows point from the text "Do ukończenia konfiguracji Cypressa warto w folderze z projektem utworzyć plik .prettierrc, a także podstawowa konfiguracja w pliku cypress.json. W folderze z Cypressem plik tsconfig.json - pliki będą dostępne na moim githubie - > https://github.com/sebb94/cypress_webinar" to the **cypress.json** file in the Explorer and the **cypress.json** tab in the Open Editors bar.

5. Struktura Cypressa

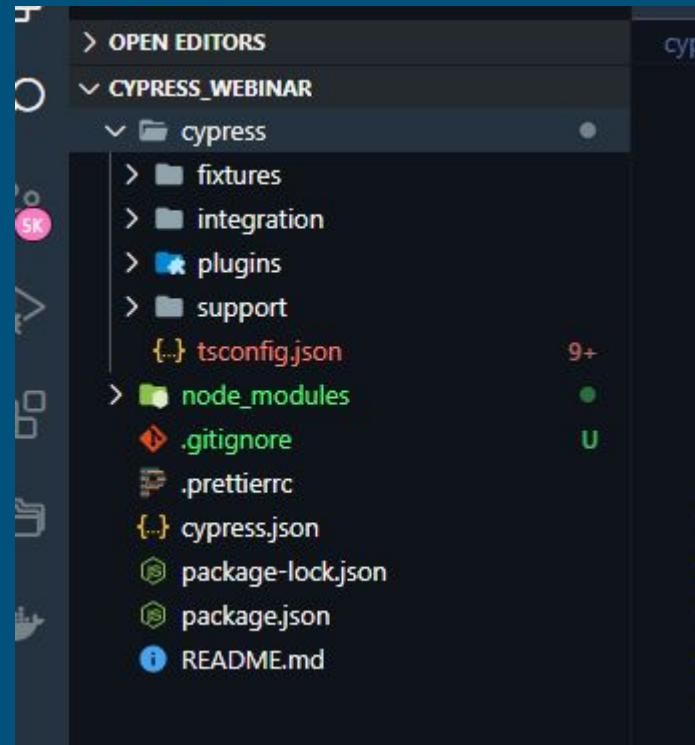
Cypress - główny folder z Cypressem

Cypress/fixtures - folder z plikami .json z przykładowymi danymi

Cypress/integration - folder z testami

Cypress/plugins - tutaj importuje się dodatkowe pluginy

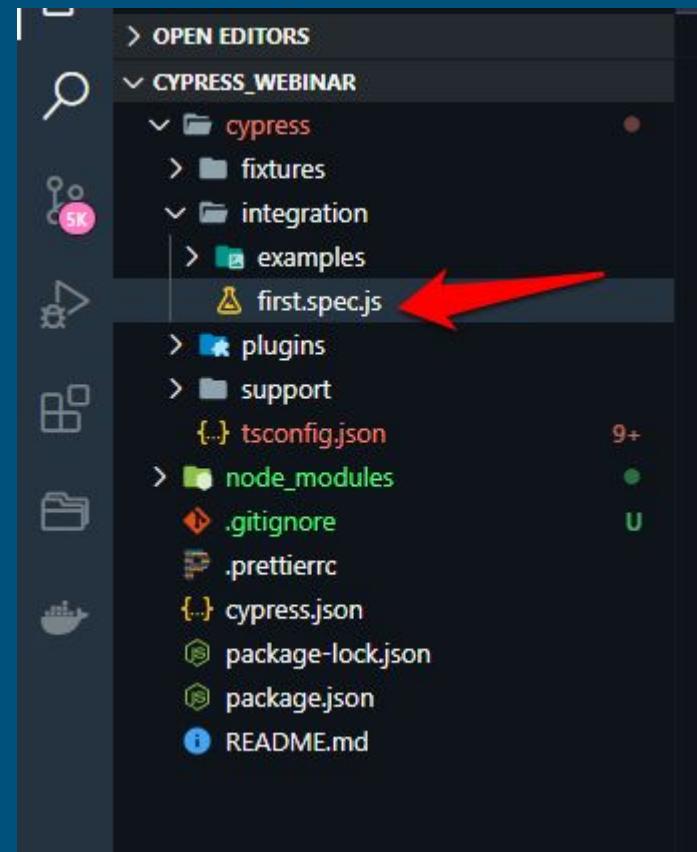
Cypress/support - zawiera np. nasze customowe komendy



6. Struktura testu - Mocha

Cypress "zapozyczył" strukturę pliku testowego z Mocha framework - głównymi składowymi pliku spec.js są describe() oraz it() -> skrót od individual test.

Stwórzmy wreszcie jakiś test - będzie to plik first.spec.js w folderze integration



Stworzyliśmy describe() oraz 3 testy - funkcja cy.log() służy do wyświetlania tekstu

Wykonajmy w terminalu komendę

npm run cy:open

I wykonajmy nasz test

```
cypress > integration > first.spec.js > describe('My first test') callback > it('My second test step') callback
1  describe('My first test', () => {
2
3    it('My first test step', () => {
4      cy.log('1st step')
5    });
6
7    it('My second test step', () => {
8      cy.log(['2nd step'])
9    );
10
11   it('My third test step', () => {
12     cy.log('3rd step')
13   );
14
15 });


```

C:\Users\Sebastian\Desktop\cypress\cypress_webinar

File Edit View Window Help

cypress_webinar

Support Docs Log In

Tests Runs Settings Stop Running Chrome 86

Search... Running 1 spec

INTEGRATION TESTS COLLAPSE ALL EXPAND ALL

examples first.spec.js

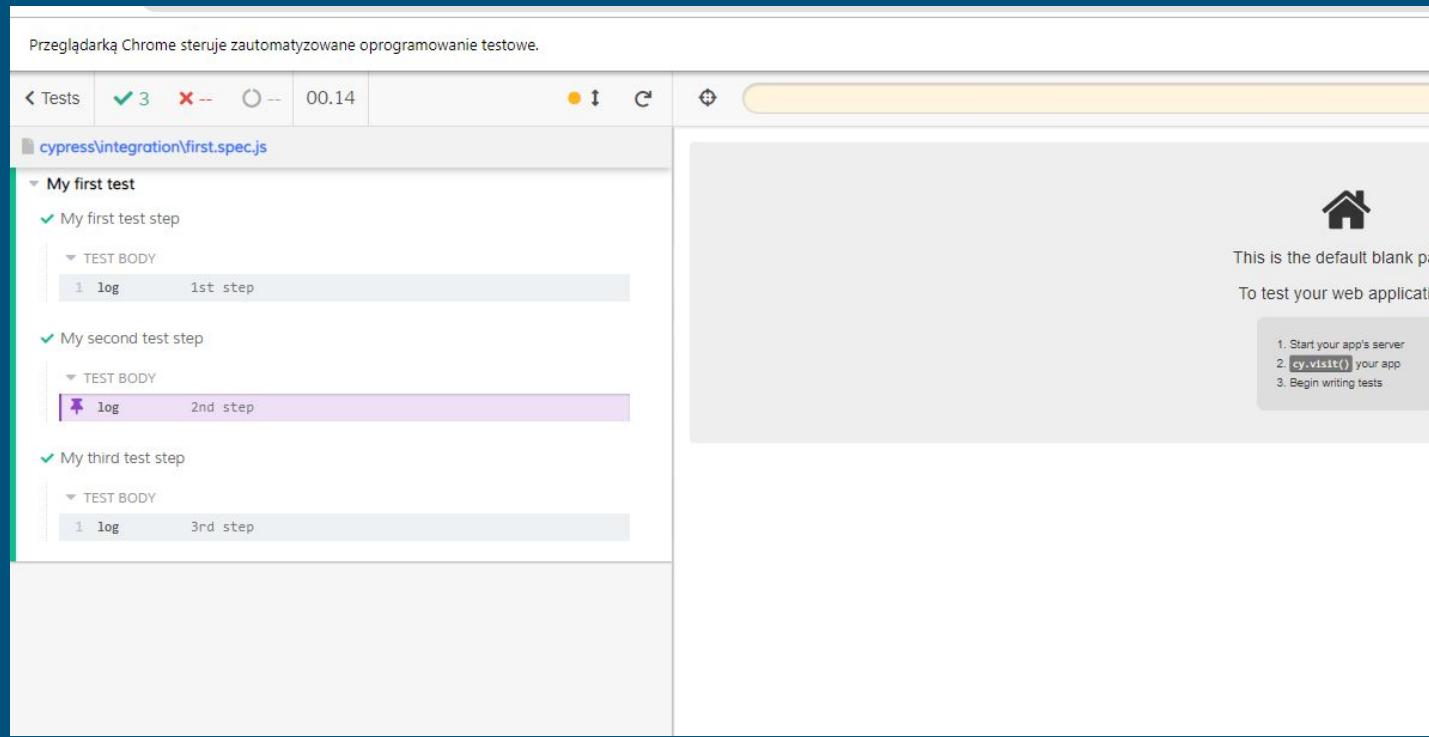
tutuaj wybieramy test, który chcemy uruchomic

tu wybieramy przeglądarkę

Version 5.5.0 Changelog

The screenshot shows the Cypress UI interface. At the top, there's a navigation bar with links for File, Edit, View, Window, Help, Support, Docs, and Log In. Below the navigation bar is a toolbar with buttons for Tests (selected), Runs, Settings, Stop (red button), and Running Chrome 86. A search bar is present. On the left, a sidebar lists 'INTEGRATION TESTS' with options to 'COLLAPSE ALL' or 'EXPAND ALL'. Underneath, there are two items: 'examples' and 'first.spec.js', with 'first.spec.js' currently selected. Two red arrows point from the text annotations to these elements: one arrow points to 'first.spec.js' with the text 'tutuaj wybieramy test, który chcemy uruchomic', and another arrow points to the 'Running Chrome 86' status with the text 'tu wybieramy przeglądarkę'.

Nasz test wykonał się, poprawnie wyświetlając tekst na ekranie



8. Hooki - pomocna dłoń

Hooki pomagają wykonywać akcje w określonym czasie - głównie pomagają unikać nam duplikacji kodu, który chcemy np wykonać przed każdym testem indywidualnym -> it()

before() -> hook ten wykonuje się raz, przed wszystkimi it()

beforeEach() -> wykonuje się przed każdym it()

afterEach() -> wykonuje się po każdym it()

after() -> wykonuje się po tym, jak ostatni it() w test suicie zostanie wykonany

Oto rezultat po wykonaniu testu z naszymi hookami - zgodnie z założeniami before() i after() wykona się raz, a beforeEach() i afterEach() 3 razy, ponieważ tyle mamy indywidualnych testów it()

Założymy sytuację, że mamy 20 it() w pliku spec.js a chcemy wykonać tylko 1 lub 2 z nich, lub analogicznie z 20 wykonać 20 a pominąć tylko dwa - czy w takim przypadku musimy skomentować te it(), których nie chcemy wykonywać ? Absolutnie nie - z pomocą przychodzą kolejne hooki - **skip()** i **only()**

The screenshot shows a test runner interface with three test steps:

- My first test**:
 - My first test step**:
 - BEFORE ALL**: 1 log Ja wykonam się raz!
 - BEFORE EACH**: 1 log Ja z tego co wiem wykonam się 3 razy
 - TEST BODY**: 1 log 1st step
 - AFTER EACH**: 1 log Ja też 3 razy się wykonam - ale trochę się spóźnię

My second test step:
 - BEFORE EACH**: 1 log Ja z tego co wiem wykonam się 3 razy
 - TEST BODY**: 1 log 2nd step
 - AFTER EACH**: 1 log Ja też 3 razy się wykonam - ale trochę się spóźnię

My third test step:
 - BEFORE EACH**: 1 log Ja z tego co wiem wykonam się 3 razy
 - TEST BODY**: 1 log 3rd step
 - AFTER EACH**: 1 log Ja też 3 razy się wykonam - ale trochę się spóźnię
 - AFTER ALL**: 1 log Halo? Jest tu kto?

```
11    afterEach(() => {
12      cy.log("Ja też 3 razy się wykonam - ale trochę się
13    });
14
15    after(() => {
16      cy.log("Halo? Jest tu kto?")
17    });
18
19    it('My first test step', () => {
20      cy.log('1st step')
21    );
22
23    it('My second test step', () => {
24      cy.log('2nd step')
25    );
26
27    it.only('My third test step', () => {
28      cy.log('3rd step')
29    );
30  );
31}
```

The screenshot shows the Cypress Test Runner interface. A red arrow points from the 'it.only' line in the code to the 'My third test step' section in the runner. The 'My first test' suite is expanded, showing the following steps:

- My third test step** (green checkmark):
 - BEFORE ALL**: 1 log Ja wykonam się raz!
 - BEFORE EACH**: 1 log Ja z tego co wiem wykonam się 3 razy
 - TEST BODY**: 1 log 3rd step
 - AFTER EACH**: 1 log Ja też 3 razy się wykonam - ale trochę się spóźnie
 - AFTER ALL**: 1 log Halo? Jest tu kto?

Jak widzimy wykonał się tylko 3 it() - czyli zgodnie z założeniem - również nasze hooki, które normalnie wykonują się 3 razy - tym razem wszystkie wykonały się raz - zgodnie z kolejnością

```
5  after(() => {
6    |   cy.log("Halo? Jest tu kto?")
7  );
8
9  it('My first test step', () => {
10   |   cy.log(['1st step'])
11 });
12
13 it.skip('My second test step', () => {
14   |   cy.log('2nd step')
15 });
16
17 it('My third test step', () => {
18   |   cy.log('3rd step')
19 });
20
21 });
```

The screenshot shows the Cypress Test Runner interface with the following details:

- Test File:** cypress/integration/first.spec.js
- Test Suite:** My first test
- Test Step:** My first test step
- Logs:**
 - BEFORE ALL: 1 log Ja wykonam się raz!
 - BEFORE EACH: 1 log Ja z tego co wiem wykonam się 3 razy
 - TEST BODY: 1 log 1st step
 - AFTER EACH: 1 log Ja też 3 razy się wykonam - ale trochę się spóźnione
 - AFTER ALL: 1 log Halo? Jest tu kto?
- Test Step:** My second test step (skipped)
- Logs:**
 - BEFORE EACH: 1 log Ja z tego co wiem wykonam się 3 razy
 - TEST BODY: 1 log 3rd step
 - AFTER EACH: 1 log Ja też 3 razy się wykonam - ale trochę się spóźnione
 - AFTER ALL: 1 log Halo? Jest tu kto?
- Test Step:** My third test step
- Logs:**
 - BEFORE EACH: 1 log Ja z tego co wiem wykonam się 3 razy
 - TEST BODY: 1 log 3rd step
 - AFTER EACH: 1 log Ja też 3 razy się wykonam - ale trochę się spóźnione
 - AFTER ALL: 1 log Halo? Jest tu kto?

W przypadku "skip" - cypress runner poinformował nas o tym, że pominęliśmy dany it() za pomocą odpowiedniej ikonki oraz jaśniejszego tekstu przy nazwie it()

9. Sposoby wybierania elementów - przypominamy sobie cssa :)

Zanim przejdę do omówienia praktycznych testów - chciałbym omówić funkcję **cy.get()** -> wybiera ona interesujący nas element html na stronie. Do wybierania elementów do wykonania akcji w teście służy nic innego jak selektor css - na identycznej zasadzie jak w jQuery funkcja `$(‘selector’)`. Utworzyłem plik `selectors.spec.js` gdzie pokazałem kilka przykładowych selektorów

cypress > integration > selectors.spec.js > describe("Przykłady selektorów") callback > it("Przykłady selektorów") callback

```
1  describe("Przykłady selektorów", () => {
2    it("Przykłady selektorów", () => [
3      //By tag name
4      cy.get("input")
5      // By attribute
6      cy.get("[placeholder]")
7      //By attribute name and value
8      cy.get("input[name='first_name']")
9      //By id
10     cy.get("#contact_me")
11     //By class
12     cy.get(".feedback-input")
13     //By multiple classes
14     cy.get("[class='navbar navbar-inverse navbar-fixed-top']")
15     //By two different attributes
16     cy.get("[name='email'][placeholder='Email Address']")
17     // By tag name, attribute with value, Id and Class name
18     cy.get('input[placeholder="Email"]#inputEmail.input-full-width')
19   ])
20 })
```

10. Test formularza kontaktowego

Plik **form.spec.js**

Założenia testu:

1. Załóż stronę kontaktową
2. Zaznacz tylko zgodę do wysyłania wiadomości, pozostaw wszystkie inne pola puste
3. Zweryfikuj, czy wyświetla się ogólny komunikat o błędzie oraz komunikat o błędzie pod każdym inputem
4. Wypełnij wszystkie pola poprawnie
5. Zweryfikuj, czy wyświetla się wiadomość z podziękowaniem za wiadomość

```
describe('Should sent contact message', () => {
  it('should visit page', () => {
    cy.visit('https://tertius.pl/kontakt')
    cy.url().should('include','/kontakt')
  });
});
```

W pierwszym it() ładowamy stronę za pomocą funkcji cy.visit(), a także pobieramy url z przeglądarki za pomocą cy.url() oraz sprawdzamy za pomocą funkcji should() czy adres url zawiera frazę/kontakt - w przeciwnym wypadku test pokaże błąd

J),

```
it('should incorrect fill form', () => {
  cy.get('.preloader').invoke('css', 'display', 'none')
  cy.contains('Zgoda').click()
  cy.wait(1000)
  cy.get('.acceptance-221 input[type="checkbox"]').check()
  cy.contains('Wyślij wiadomość').click()
});
```

Podczas tworzenia tego kroku napotkałem problem, że preloader nie chciał zniknąć, uniemożliwiając dalsze wykonanie testu - czasami działało czasami nie - tutaj można zobaczyć moc Cypressa - działa on bezpośrednio w przeglądarce i mamy pełną możliwość manipulowania DOMem -> w tym przypadku użyłem funkcji invoke, która jako 1 parametr przyjmuje jedną z funkcji jQuery -> np css() lub removeAttr() i ukryłem preloader aby mieć 100% pewności, że preloader nie wypatrzy wyniku testu.

cy.contains('Zgoda').click() -> klik na przycisk akceptujący ciasteczkę

cy.wait(1000) -> czekamy 1 sekundę

Następnie za pomocą funkcji .check() -> zaznaczamy zgodę RODO do wysłania wiadomości

Oraz na końcu klikamy w przycisk który zawiera tekst "Wyślij wiadomość"

Oto spodziewany wynik naszego działania.

W kolejnym kroku sprawdzamy, czy wyświetla się dokładnie 4 elementy o błędzie oraz 1 główny komunikat

The screenshot shows a contact form with several fields and error messages:

- Imię**: Error message: "Wymagane jest wypełnienie tego pola."
- E-mail**: Error message: "Wymagane jest wypełnienie tego pola."
- Temat**: Error message: "Wymagane jest wypełnienie tego pola."
- Wiadomość**: Error message: "Wymagane jest wypełnienie tego pola."
- Consent checkbox**: Description: "Zgadzam się na przetwarzanie moich danych osobowych (imię, nazwisko, adres email) przez firmę "TERTIUS - Sebastian Gągor" w celu uzyskania odpowiedzi na moje pytanie zadane przez formularz kontaktowy. Wyrażenie zgody jest dobrowolne. Mam prawo cofnięcia zgody w dowolnym momencie bez wpływu na zgodność z prawem przetwarzania, którego dokonano na podstawie zgody przed jej cofnięciem. Mam prawo dostępu do treści swoich danych i ich sprostowania, usunięcia, ograniczenia przetwarzania, oraz prawo do przenoszenia danych na zasadach zawartych w polityce prywatności strony internetowej. Dane osobowe na stronie internetowej są zgodnie z Polityką Prywatności. Proszę zapoznać się z polityką przed wyrażeniem zgody."
- Send button**: "WYSJ WIADOMOŚĆ"

A yellow-bordered box at the bottom contains the text: "Przynajmniej jedno pole jest błędnie wypełnione. Sprawdź wpisaną treść i spróbuj ponownie."

```
});  
  
it('should error message be visible', () => {  
    cy.contains('Przynajmniej jedno pole jest błędnie wypełnione. Sprawdź wpisaną treść i spróbuj ponownie.')  
        .should('be.visible')  
    cy.get('.wpcf7-not-valid-tip').should('have.length', 4)  
});
```

Następnie wypełniamy prawidłowo formularz - funkcja type() wpisuje w inputy tekst podany w parametrze. W ostatnim kroku walidujemy, czy widoczna jest wiadomość z podziękowaniem za wysłanie wiadomości

```
it('should correct fill form', () => {
    cy.get('input[placeholder="Imię"]').type("Testowy")
    cy.get('input[placeholder="E-mail"]').type("gagor.sebastian@gmail.com")
    cy.get('input[placeholder="Temat"]').type("Testowy temat")
    cy.get('[placeholder="Wiadomość"]').type("Testowa wiadomość")
    cy.get('.acceptance-221 input[type="checkbox"]').check()
    cy.contains('Wyślij wiadomość').click()
});

it('should thank you message be visible', () => {
    cy.wait(1000)
    cy.contains('Twoja wiadomość została wysłana. Dziękujemy!').should('be.visible')
});
```

Imię

E-mail

Temat

Wiadomość

zgadzam się na przetwarzanie moich danych osobowych (imię nazwisko, adres email) przez firmę "TERTIUS - Sebastian Gagor" w celu uzyskania odpowiedzi na moje pytanie zadane przez formularz kontaktowy. Wyrażenie zgody jest dobrowolne. Mam prawo cofnięcia zgody w dowolnym momencie bez wpływu na zgódność z prawem przetwarzania, którego dokonano na podstawie zgody przed jej cofnięciem. Mam prawo dostępu do treści swoich danych i ich sprostowania, usunięcia, ograniczenia przetwarzania, oraz prawo do przenoszenia danych na zasadach zawartych w polityce prywatności strony internetowej. Dane osobowe na stronie internetowej są zgodnie z Polityką Prywatności. Proszę zapoznać się z polityką przed wyrażeniem zgody.

WYSZLI WIADOMOSC

Twoja wiadomość została wysłana. Dziękujemy!

Wykonajmy nasz test

Przeglądarka Chrome steruje zautomatyzowane oprogramowanie testowe.

< Tests	✓ 5	✗ --	⌚ -	08.47
---------	-----	------	-----	-------

https://tertius.pl/kontakt/

cypress/integration/form.spec.js

- Should sent contact message
 - ✓ should visit page
 - ✓ should incorrect fill form
 - ✓ should error message be visible
 - ✓ should correct fill form
 - ✓ should thank you message be visible

Zgadzam się na przetwarzanie moich danych osobowych (imię, nazwisko, adres email) przez firmę "TERTIUS - Specjalistyczny Gabinet Wspierania i Rozwiązywania Problemów" na cele pytanie zadane na stronie. Przetwarzanie danych jest jednoznaczne. Mam pełną cofnięcię zgodę w任何时候 do zmiany swojej zgody bez wpływu na zgodność z prawem przetwarzania, którego dokonano na podstawie zgody przed jej cofnięciem. Mam prawo dostępu do treści swoich danych i ich sprostowania, usunięcia, ograniczenia przetwarzania, oraz prawo do przenoszenia danych na zasadach zawartych w polityce prywatności strony internetowej. Dane osobowe na stronie internetowej są zgodnie z Polityką Prywatności. Proszę zapoznać się z polityką przed wyrażeniem zgody.

WYSŁIJ WIADOMOŚĆ

Twoja wiadomość została wysłana. Dziękujemy!

**TERTIUS**
Firma zajmuje się tworzeniem

Menu
»  Strona główna
» O Nas
» Sklep

Uslugi
»  Zarządzanie projektami
»  Systemy informacyjne
»  Systemy zarządzania
»  Systemy dla użytkowników

Mail dotarł

The screenshot shows an email client interface with a dark theme. At the top, there's a header bar with the title "Tertius - "Testowy temat"" and an arrow pointing to "biuro@tertius.pl". To the right of the arrow is the time "20:49". Below the header are several action buttons: "Odpowiedz" (Reply), "Przekaż" (Forward), "Archiwizuj" (Archive), "Niechciana" (Unsubscribe), "Usuń" (Delete), and "Więcej" (More). The main content area displays an incoming email message. The message is from "Ja <biuro@tertius.pl>" with the subject "Tertius - "Testowy temat"". It has a reply-to address "Odp. do Ja <gagor.sebastian@gmail.com>" and a reply抄送地址 "Do Ja <biuro@tertius.pl>". The message body contains the text "Nadawca: Testowy gagor.sebastian@gmail.com" and "Temat: Testowy temat". Below this, it says "Treść wiadomości:" followed by "Testowa wiadomość". At the bottom, a note states "Ta wiadomość została wysłana przez formularz kontaktowy na stronie Tertius (<https://tertius.pl>)."

W tym miejscu chciałbym zaprezentować funkcjonalność Cypressa, która wyróżnia go spośród innych narzędzi do tworzenia testów automatycznych - time traveller. Cypress po każdej akcji robi screenshoty, aby dać nam możliwość łatwego debugowania i sprawdzania, co dzieje się po kolei w DOMie-- mamy dostępny stan "przed" i "po" - idealnym przykładem będzie tutaj krok z wpisaniem wartości do inputa. Mamy stan "before" przed wpisaniem imienia - widzimy pusty input

The screenshot shows the Cypress Test Runner interface on the left and a browser screenshot on the right.

Cypress Test Runner (Left):

- Tests: 5 passed, 0 failed, 0 pending.
- File: cypress/integration/Form.spec.js
- Test: Should sent contact message
- Code Snippet:

```
1 get      input[placeholder="Imię"]
2 - type  Testowy
3 get      input[placeholder="E-mail"]
4 - type  gagar.sebastian@gmail.com
5 get      input[placeholder="Temat"]
6 - type  Testowy temat
7 get      [placeholder="Wiadomość"]
8 - type  Testowa wiadomość
9 get      .acceptance-221 input[type="checkbox"]
10 - check Wyślij wiadomość
11 contains Wysłaj wiadomość
12 - click
(xhr)    POST 200 /wp-json/contact-form-7/v1/contact-forms/2...
```
- Test: Should thank you message be visible

Browser Screenshot (Right):

The browser displays the TERTIUS website. A red arrow points to the 'Imię' input field, which is empty. Another red arrow points to the 'E-mail' input field, which also appears empty. A third red arrow points to the 'Temat' input field, which is empty. A fourth red arrow points to the 'Wiadomość' text area, which is empty. A fifth red arrow points to a yellow warning box at the bottom stating: "Przy najmniej jedno pole jest błędnie wypełnione. Sprawdź wpisaną treść i spróbuj ponownie." A sixth red arrow points to the bottom navigation bar where the 'DOM Snapshot (pinned)' button is highlighted, and the 'before' and 'after' buttons are visible.

Oraz stan "after" - widzimy tutaj już wpisaną wartość "testowy" do inputa z imieniem

- ✓ should incorrect fill form
- ✓ should error message be visible
- ✓ should correct fill form

▼ TEST BODY

```
1 get      input[placeholder="Imię"]  
2 - type   Testowy  
3 get      input[placeholder="E-mail"]  
4 - type   gagar.sebastian@gmail.com  
5 get      input[placeholder="Temat"]  
6 - type   Testowy temat  
7 get      [placeholder="Wiadomość"]  
8 - type   Testowa wiadomość  
9 get      .acceptance-221 input[type="checkbox"]  
10 - check  
11 contains Wyślij wiadomość  
12 - click  
(xhr)     ● POST 200 /wp-json/contact-form-7/v1/contact-forms/2...
```

- ✓ should thank you message be visible

The screenshot shows a contact form with several fields and validation messages. A red arrow points from the test step '2 - type Testowy' to the 'Imię' field, which now contains the value 'Testowy'. Another red arrow points to the 'E-mail' field, which has the placeholder 'gagar.sebastian@gmail.com'. A third red arrow points to the 'Temat' field, which has the placeholder 'Testowy temat'. A fourth red arrow points to the 'Wiadomość' field, which has the placeholder 'Testowa wiadomość'. A yellow box at the bottom displays the message: 'Przynajmniej jedno pole jest błędnie wypełnione. Sprawdź wpisaną treść i spróbuj ponownie.' A red arrow points to the 'WYŚLIJ WIADOMOŚĆ' button. At the bottom, a 'DOM Snapshot (pinned)' button is shown with 'before' and 'after' tabs, with a red arrow pointing to the 'after' tab.

11. Smoke testy nawigacji

W tym teście naszym celem jest sprawdzanie, czy podstrony istnieją i wyświetla się błąd 404 czy 500. Funkcja acceptCookies() pozwala nie powielać kodu do akceptacji ciasteczek. Zostały również ukazane asserty typu Chai-jquery, które również są dostępne w Cypressie

```
cy.visit('https://tertius.pl')  
cy.url().should('include','tertius.pl')  
acceptCookies()  
cy.contains('h2','Zacznij swój biznes w internecie już dziś!').should('be.visible')  
expect(true).to.be.equal(true)  
expect(3).to.be.greaterThan(2)  
});  
  
it('should visit about-us page', () => {  
  cy.visit('https://tertius.pl/o-nas')  
  acceptCookies()  
  cy.url().should('include','/o-nas')  
  cy.contains('h3','O nas').should('be.visible')  
});
```

Test wykonał się prawidłowo - również asserty typu chaiJquery. Tutaj znajdują się wszystkie dostępne asserty -> <https://docs.cypress.io/guides/references/assertions.html>

The image shows the Cypress Test Runner interface on the left and a browser screenshot on the right.

Cypress Test Runner (Left):

- Tests: 11 passed, 0 failed, 0 pending.
- Time: 48.81
- Spec: cypress/integration/nav.spec.js
- Test Suite: Should check pages
- Test Case: should visit home page
- Test Body (Code Snippet):

```
1 assert expected true to equal true
2 assert expected 3 to be above 2
3 visit https://tertius.pl
4 url
5 - assert expected https://tertius.pl/ to include tertius.pl
6 get .preloader
7 - invoke .css(display, none)
8 contains Zgoda
9 - click
10 contains h2, Zaczniź swój biznes w internecie już dziś!
11 - assert expected <h2> to be visible
```
- Other Test Cases:
 - should visit about-us page
 - should visit offer website page
 - should visit offer online stores page
 - should visit offer web app page
 - should visit offer mobile app page
 - should visit offer testing page
 - should visit offer wordpress plugins page
 - should visit jobs page
 - should visit blog page
 - should visit contact page

Browser Screenshot (Right):

The browser displays the Tertiust.pl contact page. The page features a large blue header with the Tertiust logo. Below the header, there's a large white input field for a message. At the bottom, there's a "Skontaktuj się" button. On the right side, there's a sidebar with a phone icon labeled "Telefon" and two phone numbers: +48 531 882 777 and +48 537 383 284.

12. Screenshoty

Za pomocą Cypressa możemy w łatwy sposób robić screenshoty na wielu podstronach oraz na wielu rozdzielczościach ekranu. Do stworzenia testu stworzę customową komendę - setResolution - w folderze support/commands.js

```
24 // -- This will overwrite an existing command --
25 // Cypress.Commands.overwrite("visit", (originalFn, url, options) => { ... })
26 cypress.Commands.add('setResolution', (size) => {
27   if (cypress._.isArray(size)) {
28     cy.viewport(size[0], size[1])
29   } else {
30     cy.viewport(size)
31   }
32 })
```

```
ESS / integration > screenshots.spec.js > describe('Screenshots') callback > sizes.forEach() callback > pages.forEach() callback > it('tu')
1 const pages = [
2   'https://tertius.pl',
3   'https://tertius.pl/kontakt' ← tablica z podstronami
4 ]
5 const sizes = ['iphone-6', 'ipad-2', [1280, 800]] ← tablica z rozmiarami
6 describe('Screenshots', () => {
7   sizes.forEach(size => {
8     pages.forEach(page => {
9       it(`#${page} --- ${size}`, () => {
10       cy.setResolution(size)
11       cy.visit(page)
12       cy.get('.preloader').invoke('css', 'display', 'none')
13       cy.contains('zgoda').click()
14       cy.wait(500)
15       cy.screenshot()
16     });
17   });
18 });
19 });
```

**ukrycie
preloadera**

funkcja do zrobienia screenshota

Testy możemy uruchamiać również z terminala - w tzw. "headless" mode:

npm cy:run -> odpala wszystkie testy z folderu integration

npm run cy:run -- --spec "cypress/integration/screenshots.spec.js" -> odpala test screenshots.spec.js

```
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:
npm ERR!     C:\Users\Sebastian\AppData\Roaming\npm-cache\_logs\2020-11-07T20_45_58_459Z-debug.log
PS C:\Users\Sebastian\Desktop\cypress\cypress_webinar> npm run cy:run -- --spec "cypress/integration/screenshots.spec.js"

> cypress_webinar@1.0.0 cy:run C:\Users\Sebastian\Desktop\cypress\cypress_webinar
> cypress run "--spec" "cypress/integration/screenshots.spec.js"

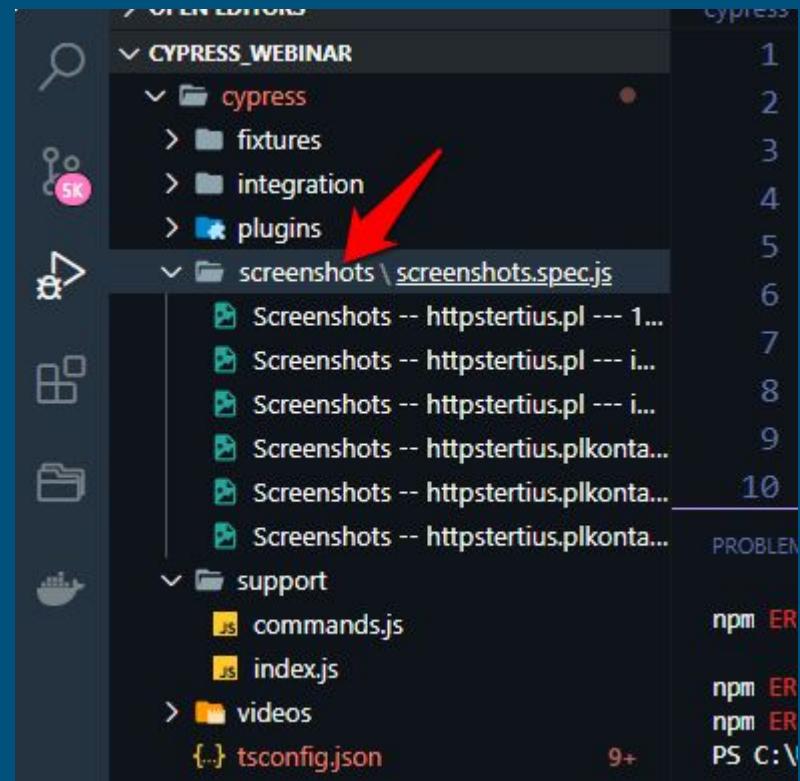
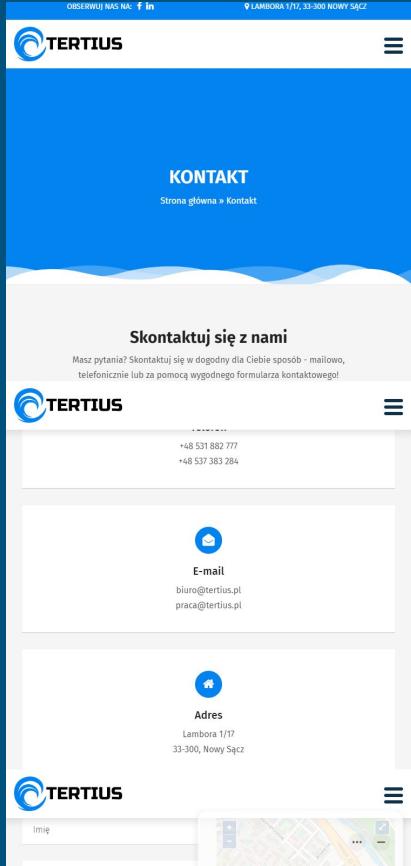
=====
(Run Starting)

Cypress: 5.5.0
Browser: Electron 85 (headless)
Specs: 1 found (screenshots.spec.js)
Searched: cypress\integration\screenshots.spec.js

=====
Running: screenshots.spec.js                               (1 of 1)

Screenshots
✓ https://tertius.pl --- iphone-6 (18898ms)
✓ https://tertius.pl/kontakt --- iphone-6 (9572ms)
✓ https://tertius.pl --- ipad-2 (16737ms)
✓ https://tertius.pl/kontakt --- ipad-2 (8292ms)
✓ https://tertius.pl --- 1280,800 (13827ms)
✓ https://tertius.pl/kontakt --- 1280,800 (9269ms)
```

W folderze screenshots znajdują się stworzone przez test screeny - w sumie 6 sztuk. Widzimy jednak problem z powielającym się sticky menu - możemy rozwiązać ten problem za pomocą znanej już nam funkcji invoke()

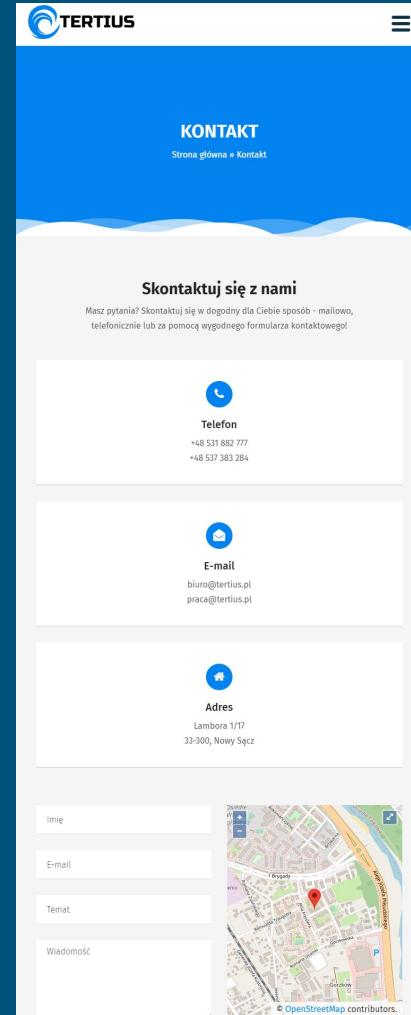


Problem ze sticky menu został rozwiązany

```
1 const pages = [
2   'https://tertius.pl',
3   'https://tertius.pl/kontakt'
4 ];
5 const sizes = ['iphone-6', 'ipad-2', [1280, 800]]
6 describe('Screenshots', () => {
7   sizes.forEach(size => {
8     pages.forEach(page => {
9       it(`${page} --- ${size}`, () => {
10      cy.setResolution(size)
11      cy.visit(page)
12      cy.get('.navbar-light').invoke('css', 'position', 'absolute')
13      cy.get('.preloader').invoke('css', 'display', 'none')
14      cy.get('#fb-root').invoke(['css', 'display', 'none'])
15      cy.contains('Zgoda').click()
16      cy.wait(500)
17      cy.screenshot()
18    });
19  });
20});
```

usuniecie sticky dla menu

↑
ukrycie messengera

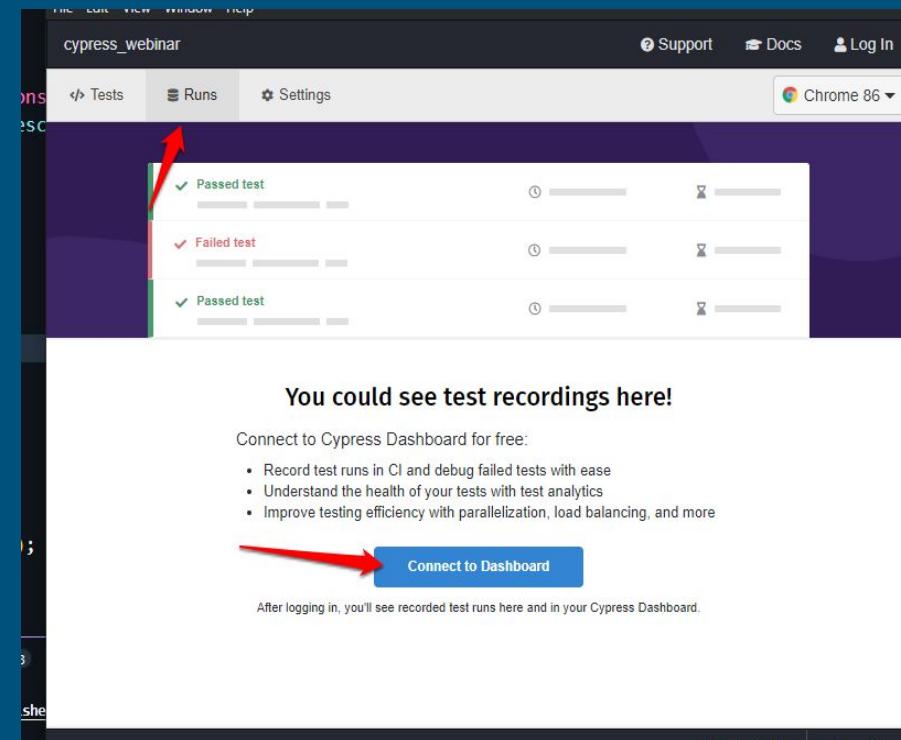


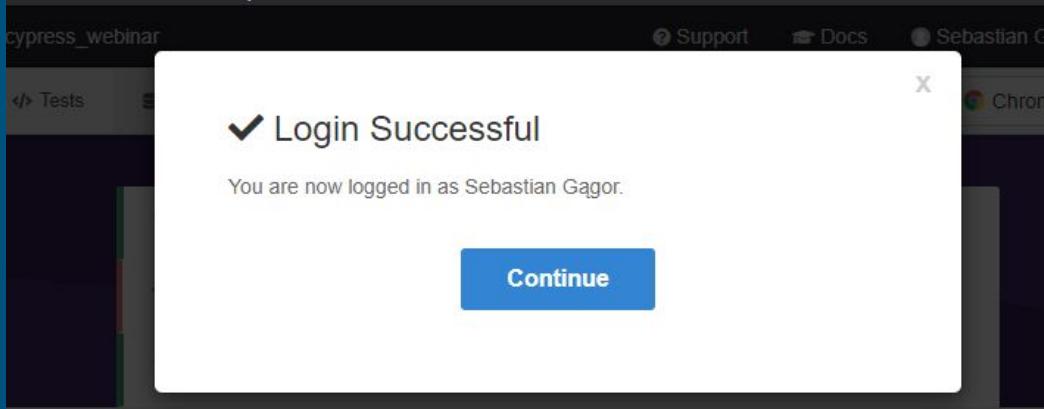
13. Cypress Dashboard

Cypress oferuje płatną funkcję, jakim jest Cypress Dashboard - do 500 it() miesięcznie jest to darmowe, więc można zobaczyć działanie dashboardu w praktyce. Więcej informacji ->

<https://www.cypress.io/dashboard/>

Aby dodać dashboard do naszego projektu należy otworzyć okienko z wyborem testu kliknąć runs -> connect to dashboard





You could see test recordings here!

Connect to Cypress Dashboard for free:

- Record test runs in CI and debug failed tests with ease
- Understand the health of your tests with test analytics
- Improve testing efficiency with parallelization, load balancing, and more

[Connect to Dashboard](#)

After logging in, you'll see recorded test runs here and in your Cypress Dashboard.

Po zalogowaniu się i stworzeniu projektu dostajemy informację, aby dodać projectId do cypress.json oraz komendę do uruchomienia testu w Dashboardzie.

The screenshot shows the Cypress Dashboard interface. At the top, there are tabs for 'Tests', 'Runs' (which is selected), and 'Settings'. A browser icon indicates 'Chrome 8'. The main content area has a heading 'To record your first run...'. It contains two numbered steps: 1. A code snippet for a cypress.json file with a projectId entry, and 2. A command to run in the CLI. A note at the bottom says recorded runs will appear in the dashboard service.

To record your first run...

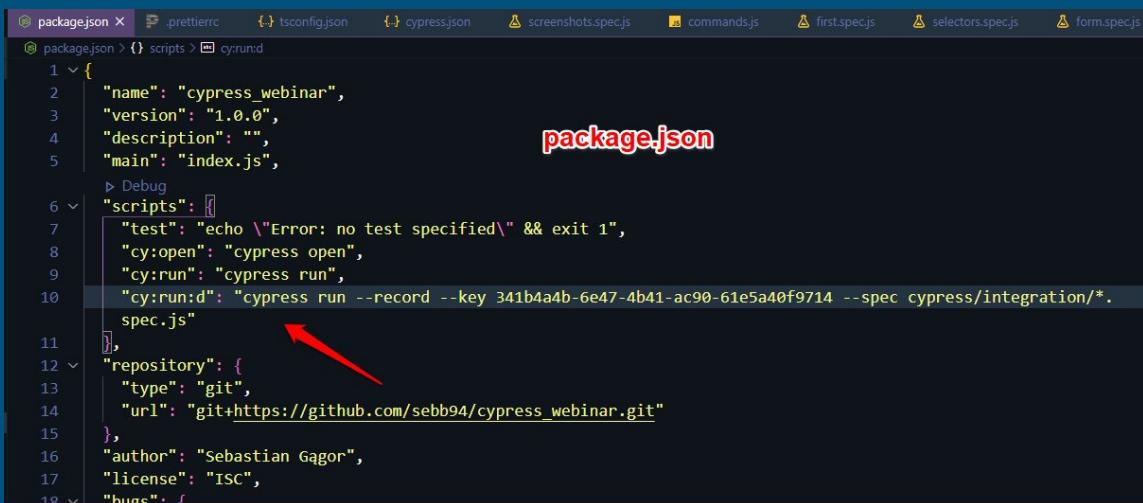
1. Check `cypress.json` file into source control.

```
1 | {  
2 |   "projectId": "jbnjps"  
3 | }
```

2. Run this command now, or in CI.

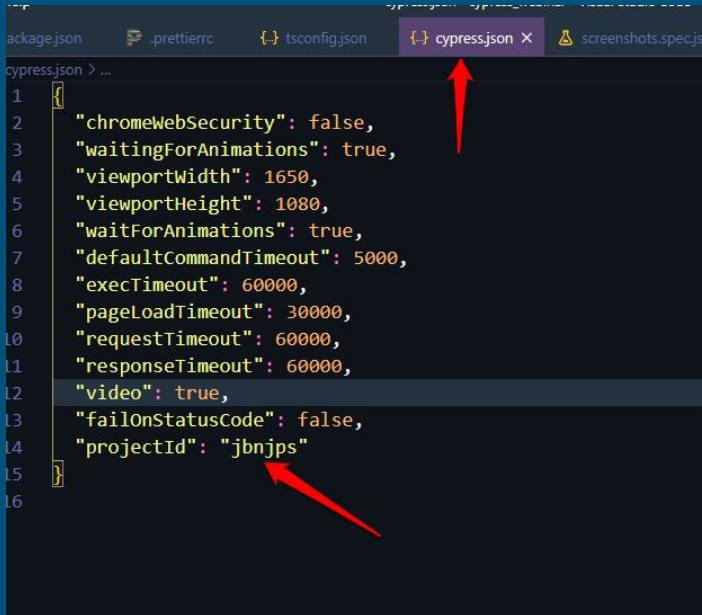
```
cypress run --record --key 341b4a4b-6e47-4b41-ac90-61e5a40f9714
```

Recorded runs will show up [here](#) and on your Cypress Dashboard Service.



```
package.json
package.json > {} scripts > cy:run:d
1  {
2    "name": "cypress_webinar",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    > Debug
7    "scripts": [
8      "test": "echo \\\"Error: no test specified\\\" && exit 1",
9      "cy:open": "cypress open",
10     "cy:run": "cypress run",
11     "cy:run:d": "cypress run --record --key 341b4a4b-6e47-4b41-ac90-61e5a40f9714 --spec cypress/integration/*.
12   ],
13   "repository": {
14     "type": "git",
15     "url": "git+https://github.com/sebb94/cypress_webinar.git"
16   },
17   "author": "Sebastian Gagor",
18   "license": "ISC",
19   "bugs": {}
```

package.json



```
cypress.json > ...
1  {
2    "chromeWebSecurity": false,
3    "waitingForAnimations": true,
4    "viewportWidth": 1650,
5    "viewportHeight": 1080,
6    "waitForAnimations": true,
7    "defaultCommandTimeout": 5000,
8    "execTimeout": 60000,
9    "pageLoadTimeout": 30000,
10   "requestTimeout": 60000,
11   "responseTimeout": 60000,
12   "video": true,
13   "failOnStatusCode": false,
14   "projectId": "jbnjps"
15 }
```

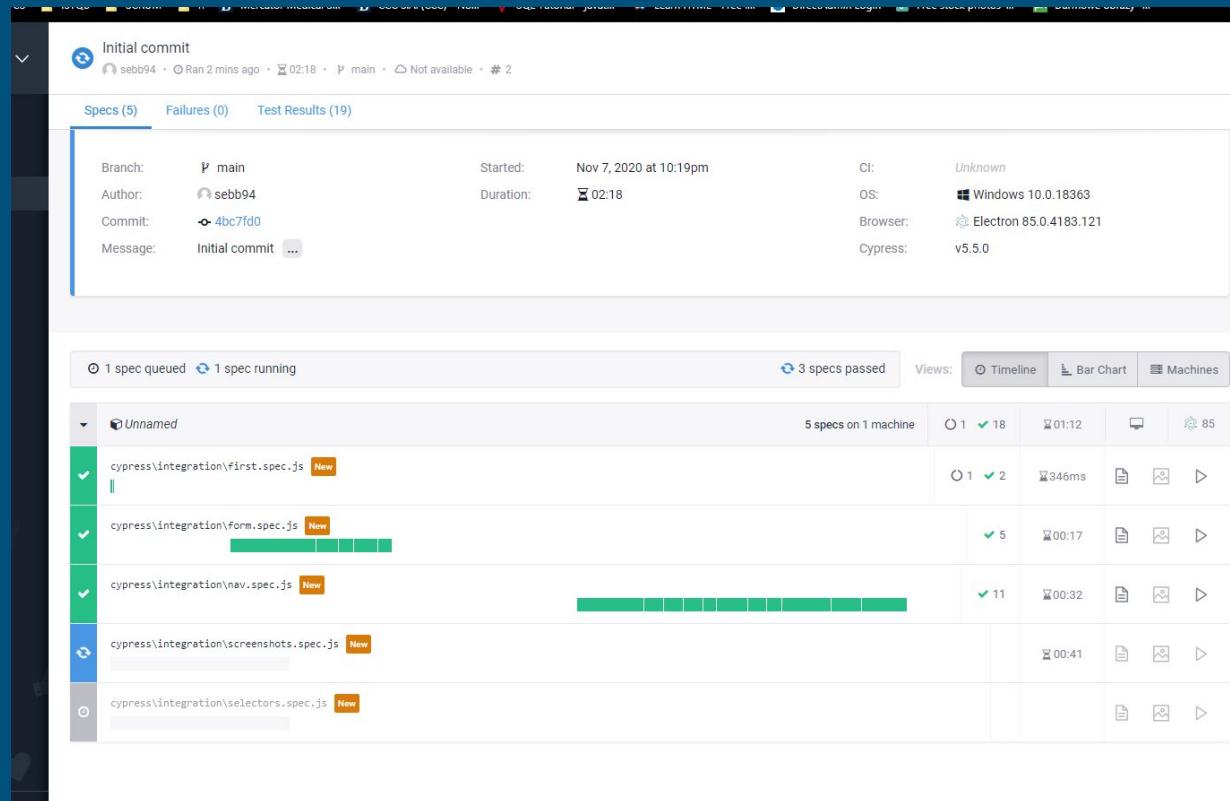
Teraz uruchomię komendę npm run cy:run:d w terminalu

```
PS C:\Users\Sebastian\Desktop\cypress_webinar> npm run cy:run:d  
> cypress_webinar@1.0.0 cy:run:d C:\Users\Sebastian\Desktop\cypress\cypress_webinar  
> cypress run --record --key 341b4a4b-6e47-4b41-ac90-61e5a40f9714 --spec cypress/integration/*.spec.js
```

(Run Starting)

```
Cypress: 5.5.0  
Browser: Electron 85 (headless)  
Specs: 5 found (first.spec.js, form.spec.js, nav.spec.js, screenshots.spec.js, selectors.spec.js)  
Searched: cypress\integration\*.spec.js  
Params: Tag: false, Group: false, Parallel: false  
Run URL: https://dashboard.cypress.io/projects/jbnjps/runs/2
```

W dashboardzie widzimy podgląd "na żywo" z przebiegu testów - dostęp do tego może mieć również nasz zespół - np. Project Manager



Jak widzimy jeden z testów zakończył się niepowodzeniem - zobaczymy dlaczego

The screenshot shows a Cypress test run interface for an 'Initial commit' by user 'sebb94'. The run was completed 5 minutes ago with a duration of 04:04. It ran on the 'main' branch and is not available. The results show 5 specs on 1 machine, with 4 passed and 1 failed.

Specs (5) Failures (1) Test Results (26)

All specs are complete! ✖ 4 specs passed ✎ 1 spec failed

Views: Timeline Bar Chart Machines

Unnamed 5 specs on 1 machine

Make your tests run 2 minutes faster by adding 3 machines in CI. See full analysis ▾ Dismiss

Spec File	Status	Time	Actions
cypress\integration\first.spec.js	Passed (New)	346ms	🔗 📈 ⌂
cypress\integration\form.spec.js	Passed	00:17	🔗 📈 ⌂
cypress\integration\nav.spec.js	Passed (New)	00:32	🔗 📈 ⌂
cypress\integration\screenshots.spec.js	Passed (New)	01:38	🔗 📈 ⌂
cypress\integration\selectors.spec.js	Failed	00:06	🔗 📈 ⌂

Red arrows point to the 'output' and 'screen' sections for the failed spec.

Running: selectors.spec.js

(5 of 5)

Przykłady selektorów

1) Przykłady selektorów

0 passing (6s)
1 failing

1) Przykłady selektorów

Przykłady selektorów:

```
AssertionError: Timed out retrying: Expected to find element: 'input', but never found it.  
at Context.eval (http://localhost:56084/_cypress/tests?p=cypress\integration\selectors.spec.js:102:8)
```

(Results)

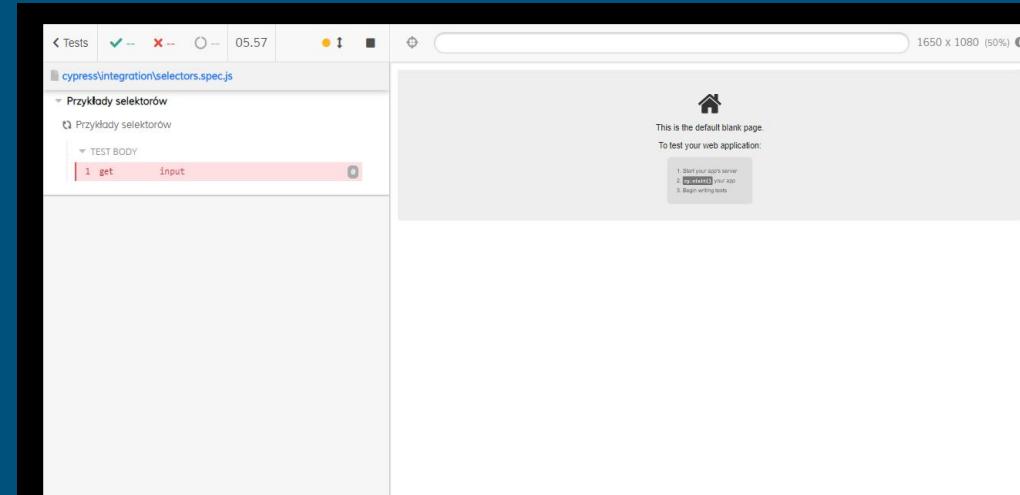
```
Tests:      1  
Passing:    0  
Failing:   1  
Pending:    0  
Skipped:   0  
Screenshots: 1  
Video:      true  
Duration:   6 seconds  
Spec Ran:   selectors.spec.js
```

1) Przykłady selektorów

Przykłady selektorów:

```
AssertionError: Timed out retrying: Expected to find element: 'input', but never found it.  
at Context.eval (http://localhost:56084/_cypress/tests?p=cypress\integration\selectors.spec.js:102:8)
```

Jak mówi nam błąd - nie odnaleziono elementu "input" - ponieważ nie dodaliśmy w tym teście funkcji cy.visit() - więc nie jest sprawdzana żadna strona internetowa



Dziękuję za uwagę :)

Przydatne linki:

https://github.com/sebb94/cypress_webinar -> link do GitHuba z materiałami z prezentacji

<https://docs.cypress.io/> -> dokumentacja Cypressa

<https://docs.cypress.io/guides/references/assertions.html> -> opisane asserty

<https://docs.cypress.io/guides/references/configuration.html#Global> -> konfiguracja cypress.json

<https://docs.cypress.io/api/commands/should.html> -> funkcja should()

<https://www.cypress.io/dashboard/> -> strona główna Dashboardu

<https://www.facebook.com/HackYeahPL/videos/2874206182863969> -> webinar

PONIŻEJ NA SLIDACH DODATKOWO - LIVE CODING ORAZ ODPOWIEDŹ NA PYTANIA Z FACEBOOKA
ORAZ WNIOSKI PO PREZENTACJI

Pytania z Facebooka

Wojciech Sygier pisze:

cy.wait() też generalnie nie powinno się używać - pomijając już fakt, że Cypress sam z siebie czeka na konkretny stan apki, to jeszcze jest niedeterministyczny (ponieważ raz coś przejdzie szybciej, raz wolniej). Jeśli już musimy czekać, to warto użyć do tego waitUntil (który można zapisać ręcznie albo pobrać plugin, który dodaje tę funkcjonalność i pisać własne waitUntil).

rozwiązanie - waitUntil wziąłem stąd: <https://github.com/NoriSte/cypress-wait-until>

I link do paczki na npm: <https://www.npmjs.com/package/cypress-wait-until>

Sebastian

Pełna zgoda - zapomniałem wspomnieć, że waits użyłem tutaj tylko po to, aby pokazać na webinarze jak Cypress kliką w button -> ponieważ robi to tak szybko, że nie możemy nawet zauważyc :) Przy porównaniu z Selenium wspominałem właśnie, że Cypress czeka na stan apki a w Selenium właśnie było to uciążliwe - tutaj plus dla twórców. Wtyczka - sprawdzę poza "wizją" jak się sprawuję -> póki co słowo do testerów manualnych, którzy chcą zacząć przygodę z automatycznymi testami - używajcie wait() do debugowania - potem należy skasować :)

Wojciech Sygier pisze:

Warto wspomnieć, że używanie after() i afterEach() to antywzorzec, ponieważ poniekąd uzależnia testy od siebie, to raz, dwa - "dangling state" po tym jak test się wykona jest bardzo przydatny przy debugowaniu.

Sebastian:

Słuszna uwaga - chciałem pokazać wszystkie hooki -> zazwyczaj używa się tylko before() - do cy.visit() jeżeli nie chcemy do tego tworzyć dodatkowego it()

Piotr Foltyn pisze:

Dwa pytania:

1) Kwestia Fixture -> Czy mozesz dać przykład jak go używać? Czy może powinno stosować się inny zamiennik do ładowania danych?

Sebastian:

Tak - pokaże na 2 przykładach - dalej w slidach będzie

Tak - pokaże na 2 przykładach

2) Czy CyPress nadaje się do testów automatycznych backendu? Ew. jakie narzędzie polecił byś do takiego

Sebastian:

Generalnie Cypress z założenia jest narzędziem do testowania frontu E2E i gotowych apek - łatwo również testować też API w Cypressie - jeżeli chodzi o backend -> rozumiem, że masz na myśli testowanie "Database testing" -> do tego Cypress nam się nie przyda -> korzystałem z tego kiedyś -> <http://jdbdt.org/> -> polcam

Robert Nord pisze

(dzisiaj mam syna pod opieką stąd od razu pytanie - czy bedziesz w stanie pod koniec podac jakies materiały do nauki cypressa? wiem, ze jest kanał oficjalny, ze jest na YT, ale moze masz cos co jest nieoczywiste w swoich rekommendacjach)

Sebastian:

Generalnie w moim przypadku było tak, że uczyłem się najpierw Selenium -> z tutoriala ->

<https://www.youtube.com/watch?v=nCJoia7wosc&list=PLhW3qG5bs-L8oRay6qeS70vJYZ3SBQnFa>

Selenium jest sporo więcej na rynku niż Cypress -> też brakowało mi tutoriala, już nawet nie mówię, żeby po polsku był -> przesiadka z Selenium (jest to kombajn) na Cypressa była dla mnie stosunkowo łatwa -> aczkolwiek poniżej podaję link do tzw "Kitchen Sink" -> mam w planach to przerobić - słyszałem opinie, że jest świetne - przykładowa apka

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

Oraz strona

<https://example.cypress.io/>

Tak jak patrzę na to - to jest to wszystko a nawet więcej, żeby pisać testy automatyczne na poziomie mida (oczywiście znając w stopniu dobrym htmla, cssa oraz jsa) -> fajnie jest to tu wy tłumaczone

Przykład fixtures nr 1.

Przykład z live coding

```
it('should go to login page', () => {
    cy.get('#signin_button').click()
});

it('should incorrect fill form', () => {
    cy.get('#user_login').type('saddsads')
    cy.get('#user_password').type('sadsadsadsadsadsa')
    cy.get('[name="submit"]').click()
});

it('should validate error message', () => {
    cy.contains('Login and/or password are wrong.').should('be.visible')
});

it('should correct fill form', () => {
    cy.get('#user_login').type('username')
    cy.get('#user_password').type('password')
    cy.get('[name="submit"]').click()
})
```

Tutaj wpisywaliśmy dane prosto w teście -> ale założymy sytuację, że sprawdzanie tego formularza jest w 20 innych plikach z testami (a tak często jest) -> jeżeli zmieni się np hasło dla niepoprawnego usera - musielibyśmy zmienić to w 20 miejscach - stąd z pomocą przychodzą fixtures .

W folderze fixtures stworzyłem plik users.json - jest to zwykły json w postaci key : value

Fixtures to dobre miejsce na nasze “dummy data”

The screenshot shows the VS Code interface with the following details:

- EXPLORER View:** Shows the project structure under "CYPRESS_WEBINAR". The "fixtures" folder contains "subjects.json" and "users.json". The "integration" folder contains "examples" (with files like "first.spec.js", "form.spec.js", "login.spec.js", "nav.spec.js", "screenshots.spec.js", and "selectors.spec.js"), "plugins", "screenshots", and "support" (with files "commands.js" and "index.js"). A red arrow points to the "examples" folder.
- Editor View:** The current file is "users.json". The code content is:

```
1  {
2    "valid_user": "username",
3    "valid_password": "password",
4    "invalid_user": "saddsadsadsaadsdsda",
5    "invalid_password": "saddsadsadsaadsdsda"
6 }
```
- Status Bar:** Shows the path "cypress > fixtures > users.json" and the file name "users.json".

```

cypress > integration > login.spec.js > describe('validate login on web zero app') callback > it('should incorrect fill form') callback
  17   // cy.get('#user_login').type('user_name')
  18   // cy.get('#user_password').type("password")
  19   cy.fixture('users').then(users => {
  20     cy.get('#user_login').type(users.invalid_user)
  21     cy.get('#user_password').type(users.invalid_password)
  22   })
  23   cy.get('[name="submit"]').click()
  24 }
  25
  26 it('should validate error message', () => {
  27   cy.contains('Login and/or password are wrong.').should('be.visible')
  28 });
  29
  30 it('should correct fill form', () => {
  31   // direct string version
  32   // cy.get('#user_login').type("username")
  33   // cy.get('#user_password').type("password")
  34   // fixtures version
  35   cy.fixture('users').then(users => {
  36     cy.get('#user_login').type(users.valid_user)
  37     cy.get('#user_password').type(users.valid_password)
  38   })
  39   cy.get('[name="submit"]').click()
  40 });
  41
  42 it('should validate profile page', () => {
  43   cy.get('.icon-user').should('be.visible')
  44   cy.contains('Balance cash accounts').should('be.visible')

```

Cy.fixture - jako argument przyjmuje nazwę jsona z folderu "fixtures" -> poprzez funkcję then możemy kontynuować nasze działania - w arrow function powtarzam nazwę fixtures ale może ona brzmieć np user -> potem w odwołaniu gdy wpiszemy users mamy nawet podpowiedź ze strony VSC

The screenshot shows a portion of a Cypress test file in VS Code. The cursor is positioned at the start of a call to `cy.fixture`. A code completion dropdown is open, listing several options:

- cy.fixture('users').then(users => {
- cy.get('#user_login').type(users. [])
- cy.get('#user_password').type(users. [])
- })
- cy.get('[name="submit"]').click()
- ;

Below the completion dropdown, the test code continues with two more `it` blocks:

- 'should validate error message', () => {
 cy.contains('Login and/or password are [] ').should('be.visible')
- 'should correct fill form', () => {
 // direct string version
 // cy.get('#user_login').type("username")
 // cy.get('#user_password').type("password")
 // fixtures version

Nasz test wykonuje się tak jak w poprzedniej wersji - teraz mamy jednak możliwość zmiany wartości w jsonie, a nie w np 20 plikach

```
validate login on web zero app
  ✓ should visit page (1665ms)
  ✓ shoud validate page heading
  ✓ should go to login page (270ms)
  ✓ should incorrect fill form (1281ms)
  ✓ should validate error message (60ms)
  ✓ should correct fill form (2261ms)
  ✓ should validate profile page (63ms)
```

7 passing (9s)

([Results](#))

Tests:	7
Passing:	7
Failing:	0
Pending:	0
Skipped:	0
Screenshots:	0
Video:	false
Duration:	8 seconds
Spec Ran:	login.spec.js

Założymy scenariusz - PM projektu dostaje codziennie z automatu maila z naszego testu formularza - nudzi go jednak czytanie wciąż tej samej wiadomości - mamy za zadanie sprawdzić, że temat wiadomości pojawiający się w mailu będzie losowy. Na początku stwórzmy jsona w folderze fixtures

The screenshot shows the VS Code interface with the file structure on the left and the content of `subjects.json` on the right. A red arrow points to the `subjects.json` file in the file tree.

```
1 {  
2     "subject_1" : "Test subject 1",  
3     "subject_2" : "Test subject 2",  
4     "subject_3" : "Test subject 3"  
5 }
```

Następnie przeróbmy nasz kod "form.spec.js" - nasze zadanie to:

- * wylosować liczbę 1 do 3
- * na tej podstawie do formularza wpisać odpowiedni temat

Ze względu na to, iż nie chciałbym stosować skomplikowanego programowania (ponieważ część widzów może znać jsa tylko w stopniu podstawowym - zastosowałem po prostu if / else -> do naszego case'a jest to dopuszczalne -> jak załącze obok mail teraz dostaje losowy temat (kolejny screen) :)

```
21
22  it('should correct fill form', () => {
23    cy.fixture('subjects').then(subject =>{
24      let randomNumber = Math.floor(Math.random() * 3) + 1;
25      cy.get('input[placeholder="Imię"]').type("Testowy")
26      cy.get('input[placeholder="E-mail"]').type("gagor.sebastian@gmail.com")
27      if(randomNumber == 1){
28        cy.get('[placeholder="Temat"]').type(subject.subject_1)
29      }else if( randomNumber == 2){
30        cy.get('[placeholder="Temat"]').type(subject.subject_2)
31      } else if (randomNumber == 3){
32        cy.get('[placeholder="Temat"]').type(subject.subject_3)
33      }
34      cy.get('[placeholder="Wiadomość"]').type("Testowa wiadomość")
35      cy.get('.acceptance-221 input[type="checkbox"]').check()
36      cy.contains('Wyślij wiadomość').click()
37    })
38  });
39});
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

/rsrc.php/v3/yL/r/N2crkBFiIy.js?_nc_x=Ij3wp8lg5Kz 200 267.708 ms --
/rsrc.php/v3/n424/vk/l/nl_P/Dw5inOFvAvF_is2_nc_x=T3Rwn81a5Kz 200 40.577 ms --

"zahardkodowane" tematy

tematy losowane z jsona

• → biuro@tertius.pl 15:27
• → biuro@tertius.pl 15:29
• → biuro@tertius.pl 19:14
• → biuro@tertius.pl 19:26
• → biuro@tertius.pl 19:26

Od Ja <biuro@tertius.pl> ☆
Temat Tertius - "Test subject 2"
Odp. do Ja <gagor.sebastian@gmail.com> ★
Do Ja <biuro@tertius.pl> ☆

Nadawca: Testowy gagor.sebastian@gmail.com
Temat: Test subject 2

Treść wiadomości:
Testowa wiadomość

--
Ta wiadomość została wysłana przez formularz kontaktowy na stronie Tertius (<https://tertius.pl>).

com jest aktualny | Nienrzeđytane: 0 | Razem: 266