

*Základy testování a učebnice  
Cypressu*

Ema Richterová

# **Obsah**

<b>ÚVOD DO TESTOVÁNÍ .....</b>	<b>4</b>
<b>TEORIE TESTOVÁNÍ .....</b>	<b>5</b>
CO JE TO TESTOVÁNÍ .....	5
QUALITY ASSURANCE VERSUS TESTOVÁNÍ SOFTWARU .....	5
MANUÁLNÍ TESTOVÁNÍ VERSUS AUTOMATIZACE .....	5
TESTOVACÍ SCÉNÁŘE .....	6
DRUHÝ TESTŮ .....	7
CO JE TO CYPRESS .....	8
<b>PRAKTIČKÁ ČÁST – TEORIE .....</b>	<b>9</b>
POTŘEBNÉ PROGRAMY A JEJICH INSTALACE .....	9
<i>Instalace Visual Studio Code .....</i>	<i>9</i>
<i>Instalace systému správy verzí kódu Gitu a NodeJS .....</i>	<i>10</i>
<i>GitHub .....</i>	<i>11</i>
<i>Instalace Cypressu .....</i>	<i>11</i>
PSANÍ CYPRESS TESTŮ .....	16
<i>Struktura testu .....</i>	<i>16</i>
<i>Komentáře v kódu .....</i>	<i>17</i>
<i>Příkazy .....</i>	<i>17</i>
<i>Vybírání prvků .....</i>	<i>18</i>
<i>Orientace v Google DevTools .....</i>	<i>20</i>
<b>PRAKTIČKÁ ČÁST – PSANÍ TESTU .....</b>	<b>24</b>
PRVNÍ TEST .....	24
DRUHÝ TEST .....	26
<b>OPRAVOVÁNÍ CHYB V KÓDU A TIPY A TRIKY PRO PSANÍ CYPRESS TESTŮ .....</b>	<b>28</b>
ZÁKLADNÍ CHYBOVÉ HLÁŠKY .....	28
DALŠÍ TIPY A TRIKY .....	30
<b>SEZNAM OBRÁZKŮ .....</b>	<b>32</b>
<b>BIBLIOGRAFIE .....</b>	<b>34</b>

## Úvod

V každé firmě, která vyvíjí nějaký software bychom našli pracovníky či týmy, jejichž členové mají názvy pozic: QA analyst, QA tester, tester, SW tester, Test Automation Engineer a mnoho dalších podobných názvů pozic. Ke 27. 12. 2023 je na portálu jobs.cz k dohledání 61 inzerátu nabízející tuto práci v Praze. Vysvětlit, co je náplní práce těchto lidí, proč existuje několik názvů na tu samou pozici a jak pracovat s frameworkm na automatizaci webových testů Cypress, najdete v této učebnici.

Všechny příklady kódu uvedené v této práci jsou dostupné v repositáři na GitHubu:

<https://github.com/StudentTraineeCenter/zaklady-testovani-Cypress>

Po naklonování složky na váš počítač a otevřením složky ve VS Code spusťte příkaz **npm install**, čímž stáhnete Cypress. Hodně štěstí!

# Teorie testování

## Co je to testování

Testování softwaru je proces v rámci kterého se snaží lidé (= testeři) najít chyby v softwaru.

Tyto chyby, nejčastěji označované anglickým slovem „bug“ (v množném čísle „bugy“), jsou testery nahlašovány a předávány vývojářům, aby je opravili. Tímto způsobem se snižuje šance, že se ke zákazníkům a koncovým klientům dostane software, který obsahuje chyby či je dokonce nefunkční.

Testování se provádí zejména ve dvou případech. V tom prvním byl architektem navržen a vývojáři naprogramován nový produkt, který je simultánně testován, aby došlo k odhalení chyb a ty byly opraveny. V druhém případě již software existuje, ale byl aktualizován. I v tomto případě dochází k testování. Někdy jsou testovány výhradně nové funkce, jindy je testována celá aplikace/program, aby se ověřilo, že nové i staré věci spolu fungují a nedošlo k narušení něčeho v minulosti fungujícího.

## Quality assurance versus testování softwaru

V oboru testování často narazíme na dva pojmy, a to *quality assurance* a *testování softwaru*. Úkolem Quality assurance engineera (inženýra) zkráceně QA engineera není pouze ověřit, jestli softwarový produkt funguje. Tento člen týmu dohlíží na celý proces vývoje.

Účastní se plánování, kontroluje, zda výsledný produkt odpovídá zadání, zda obsahuje všechny funkce, které obsahovat má. Dohlíží na jeho použitelnost z uživatelského hlediska a v neposlední řadě software otestuje. Oproti tomu se tester účastní výhradně fáze testování. (1)



Obrázek 1 – znázornění vztahu quality assurance a testování, zdroj:  
<https://kitner.cz/testovani-software/co-je-testovani-software/>

V dnešní době firmy nejčastěji poptávají QA inženýry, i když je tento název pozice volně zaměňován s titulem testera. V 99 % případů je pro zjednodušení můžeme považovat za synonyma.

## Manuální testování versus automatizace

Software se dá testovat dvěma způsoby – manuálně nebo s pomocí automatizačního nástroje. V rámci manuálního testování tester ručně prochází a proklikává jednotlivé části

aplikace a jednotlivé funkce, které obsahuje. Tento způsob testování může být zdlouhavý, nákladný na lidskou práci („člověkodny“) a je náchylnější k přehlédnutí chyby. Výhodou je, že v případě testování malých částí programu je rychlejší a nevyžaduje mnoho odborných znalostí. Právě malé požadavky na odborné znalosti bývají důvodem, proč bývá pozice manuálního testera častým způsobem, jak vstoupit do světa IT a vývoje.

Druhým způsobem, jak software otestovat, je napsat automatizované testy. Automatický test je dokument (soubor) obsahující po sobě jdoucí příkazy, jež jsou napsány jedním z programovacích jazyků. Po spuštění stou tyto příkazy vykonávány automatizačním nástrojem, jako je Cypress, Appium, Selenium nebo Playwright. Výhodou automatizovaného testování je možnost testovat kdykoliv – tedy bez ohledu na to, zda je přítomná fyzická osoba. Jakmile je test napsaný, je možné ho kdykoliv spustit a plánovat jeho spuštění dopředu.

Další výhodou je menší náchylnost k neodhalení chyby. Pokud jsou testy napsány kvalitně a pokrývají celou funkcionalitu, je při jejich spuštění důkladně otestován celý software. Nehrozí zde totiž chyb z důvodu lidského faktoru jako jsou přehlédnutí nějakého detailu.

Automatizované testy většinou prochází přes *code review* (= před použitím je kód zkонтrolován další kompetentní osobou a případně opraven), takže ho před spuštěním vidí více lidí a je vyšší šance, že se podaří odstranit nedostatky. Kdežto při manuálním testování danou funkci testuje zpravidla jeden člověk, tedy není zde chybí kontrolní síť.

Nevýhodou automatizovaného testování je nutná odborná znalost a ovládání frameworku ne-li přímo programovacího jazyka pro psaní testů.

## Testovací scénáře

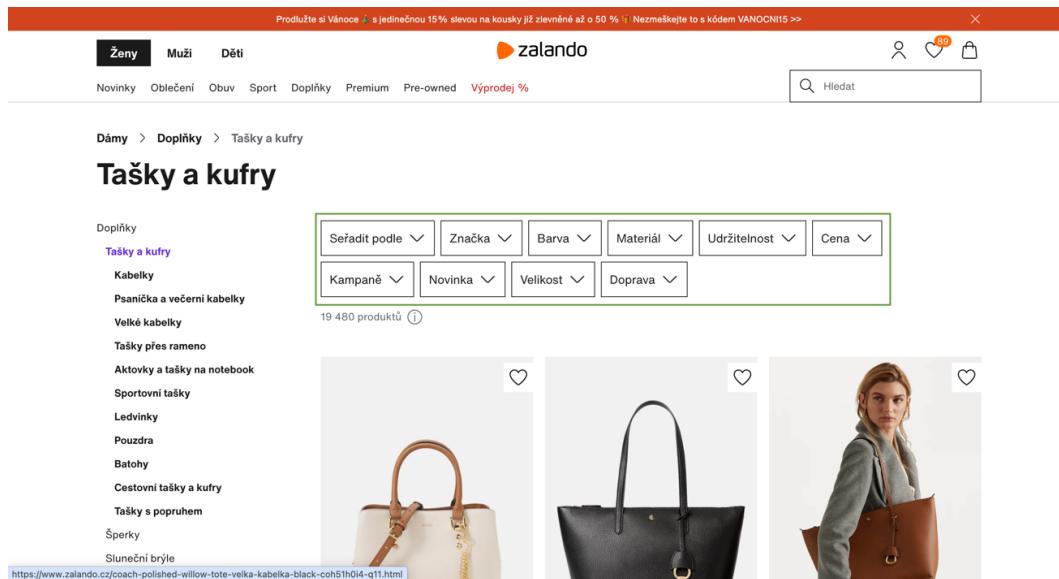
Každý test – manuální i automatizovaný – se opírá o testovací scénář (anglicky test case).

Testovací scénář je návod, jak otestovat nějakou konkrétní funkci. Skládá se ze série po sobě jdoucích kroků včetně jejich očekávaných výsledky. Pro lepší dokreslení situace scénáře často obsahují i fotky / snímky obrazovky softwaru.

Jednotlivé testovací scénáře se zaměřují i na ty nejdetailnější funkce softwaru. Pro lepší ilustraci se podíváme na konkrétní příklad – na použití filtru na webu e-shopu [Zalando](#).

Po návštěvě webové stránky tohoto e-shopu se podíváme do sekce *Doplňky* a podsekce *Tašky a batohy*. V horní části stránky nalezneme několik filtrů. Při testování těchto filtrů by

platilo, že 1 test = 1 filtr. Jeden testovací scénář by byl na otestování filtru „Seřadit podle“, jeden na „Značku“ a postupně by se postupovalo k dalším filtrům.



Obrázek 2 – e-shop Zalando, zdroj: snímek obrazovky z webu zalando.cz

Důvodem, proč se testovací scénáře píšou a používají, je, aby existoval detailní popis, jak danou funkci otestovat. Aplikace bývají často komplikované a obsahují mnoho „skrytých“ funkcí. I pro zkušeného testera je proto obtížné si zapamatovat veškeré detaile a propojené funkce, které se musí otestovat. Testovací scénáře působí jako opora. Zároveň by měly být psány tak, aby je byl i nezasvěcený uživatel schopen vykonat.

## Druhy testů

Existuje několik druhů testů, respektive účelů, kdy je používáme. Prvním druhem jsou *unit testy*, které jsou prováděny přímo vývojáři a ne testery. Programátor napíše část kódu a následně si sám napíše test, díky kterému ověří, že vše funguje, jak má. (2)

V průběhu *integračního testování* se ověřuje, jak spolu jednotlivé části aplikace, respektive jednotlivé moduly fungují a jak jsou mezi nimi přenášena data. (2) (3)

Dalším druhem jsou *end-to-end* testy zkráceně E2E testy. U tohoto typu testování se ověřují kompletní procesy jako například udělání objednávky, přidání zboží do košíku, ověření viditelnosti nějakého objektu a tak dále. E2E testy se provádějí z pohledu uživatele a k jejich automatizaci se dá například využít Cypress. (4)

*Regresivní testování* se provádí z důvodu, aby se ověřilo, že při přidání nové funkce nebyla narušena dosud fungující část aplikace či webu. (5) (2)

*Akceptační testování* se provádí těsně předtím, než software začnou využívat reální uživatelé. Ověřuje se funkčnost implementovaného systému. (6) (2) (7)

V neposlední řadě se provádí uživatelské testování neboli *usability testing*. Při tomto typu testování softwaru případně prototypu testy neprovádí tester, ale vzorek koncových uživatelů, kteří mají ověřit, že je aplikace intuitivní a snadná na používání. Při tomto typu testování je tester přítomný a zaznamenává i sleduje, jak se uživatel chová. (8) (2)

## Co je to Cypress

Cypress je nástroj pro automatizaci testování webových aplikací a webů, založený na Javascriptu. Cypress lze využít na end-to-end testování, na regresivní testování nebo na testování komponent. Psaní testů i jeho instalace je velmi jednoduchá. I proto se těší takové popularitě.

# Praktická část – teorie

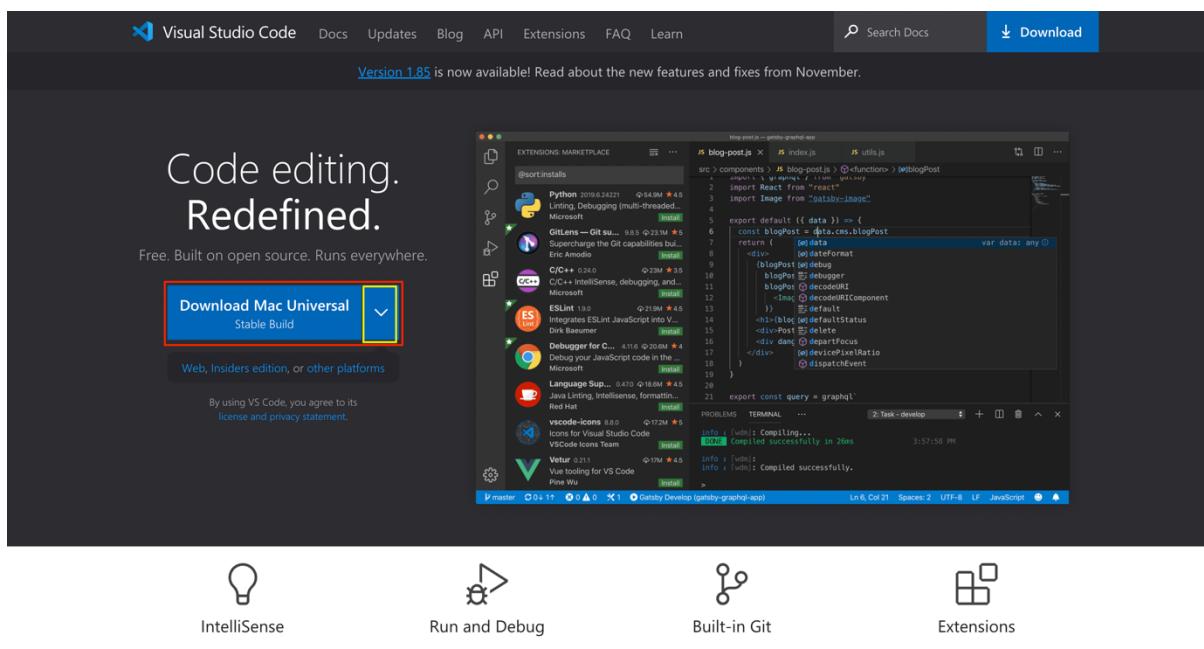
## Potřebné programy a jejich instalace

Tato práce byla vypracovaná s použitím operačního systému MacOS. Může se proto stát, že z grafického hlediska budou programy či jejich části lehce odlišné. Funkčnost tím není ovlivněna.

### Instalace Visual Studio Code

Visual Studio Code je editor kódu od společnosti Microsoft, který budeme používat v této práci a v němž budeme psát testy. Jeho instalace je velmi snadná.

1. Navštívíme web <https://code.visualstudio.com/>
2. Na titulní stránce najdeme tlačítko, kterým spustíme jeho instalaci. Stránka pozná, jaký operační systém máme a nabídne nám aktuální verzi programu.
  - a. Pokud by se ale stalo, že je systém vyhodnocen špatně, můžeme si vybrat verzi sami, a to kliknutím na žlutě vyznačený prostor.



Obrázek 3 – ukázka webu Visual Studio Code

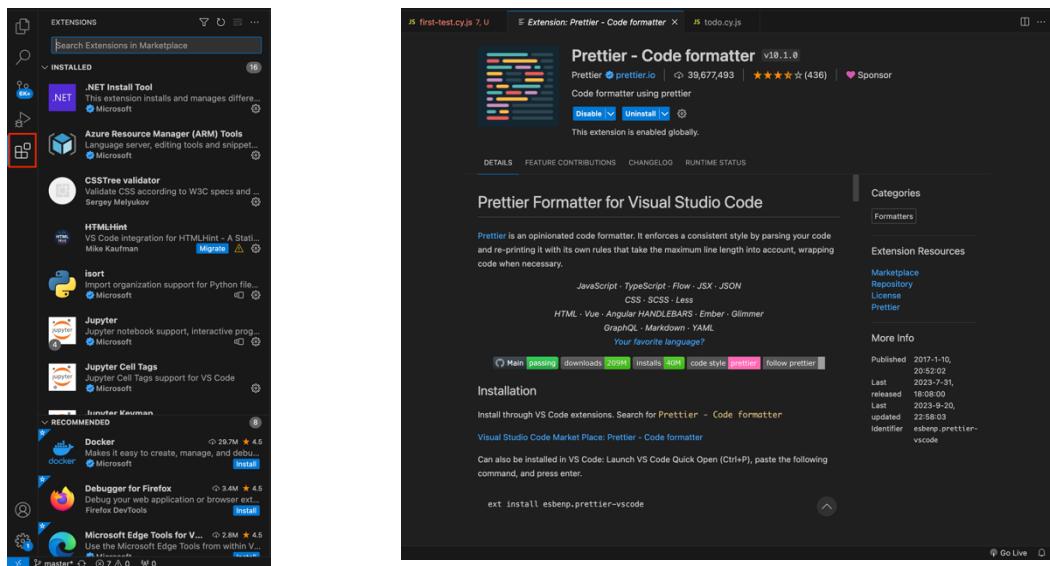
3. Po stažení se přesuneme do průzkumníku souborů, kde klikneme na stažený instalátor, čímž bude spuštěna instalace.
4. Při instalaci v systému Windows je zároveň v této fázi možné zahrnout VS Code do menu, které se zobrazí při kliknutí pravého tlačítka myši před otevřením

složky/souboru. Daný soubor se pak otevře přímo v editoru a není nutné se proklikávat přes kartu „File“.

#### Rozšíření do Visual Studio Code

Do editoru je v sekci Extensions možno stáhnout různá rozšíření.

Aby se nám s kódem v editoru lépe pracovalo, využijeme rozšíření „Prettier – Code formatter“, která nám kód pro lehčí orientaci v něm barevně a pomůže nám s formátováním. Stačí pouze kliknout na tlačítko „Install“.



Obrázek 4 – Rozšíření Prettier – Code formatter

#### Obrázek 5 – sekce Extensions

Instalace systému správy verzí kódu Gitu a NodeJS

Abychom mohli lépe pracovat s naším kódem, lépe spravovat jeho verze a ukládat si ho do GitHubu, je nutné si stáhnout systém správy verzí kódu [Git](#). Na každý operační systém se Git stahuje odlišně.

Do systému Windows je instalace velmi jednoduchá – stačí použít instalátor dostupný na webu Gitu. Instalace na MacOS je složitější, jelikož je k instalaci nutné mít nainstalovaný správce balíčků, jako například [Homebrew](#). Na YouTube existují desítky tutoriálů, jak tyto systémy stáhnout, proto autorka doporučuje zhlédnout video tutoriál.

Mimo Git je ještě potřeba stáhnout [NodeJS](#), kde najdeme pro nás důležitý balíček npm, pomocí kterého stáhneme a budeme ovládat samotný Cypress. Stejně jako u Gitu, se i NodeJS stahuje lehce rozdílně v závislosti na operačním systému. Proto i zde autorka doporučuje shlédnout tutoriál.

## GitHub

V této práci budeme dále pracovat s GitHubem – platformou poskytující hosting pro kód.

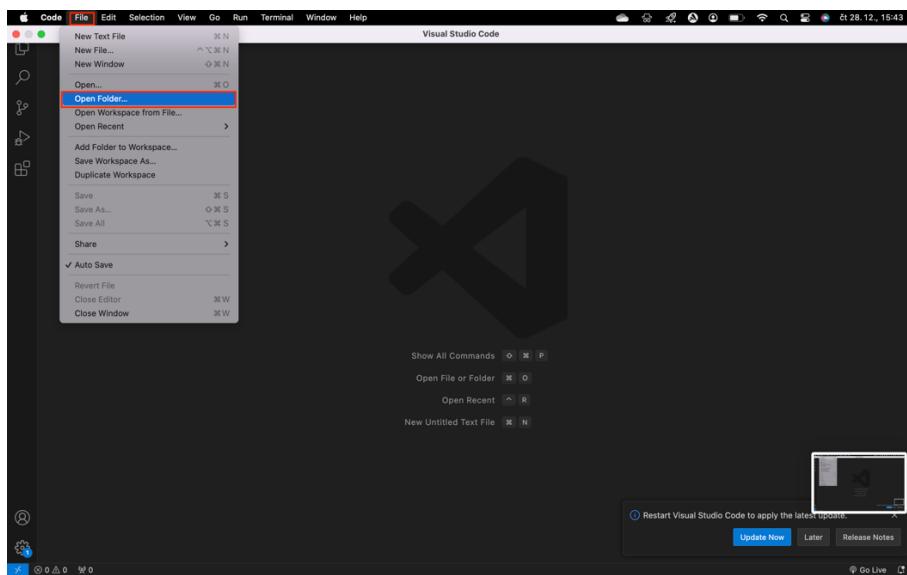
Vlastnit účet na GitHubu není nutné pro napsání Cypress testů.

Všechn kód a příklad testů, které jsou použity v této práci naleznete v repositáři uložené na této platformě na odkazu: <https://github.com/StudentTraineeCenter/zaklady-testovani-Cypress>

## Instalace Cypressu

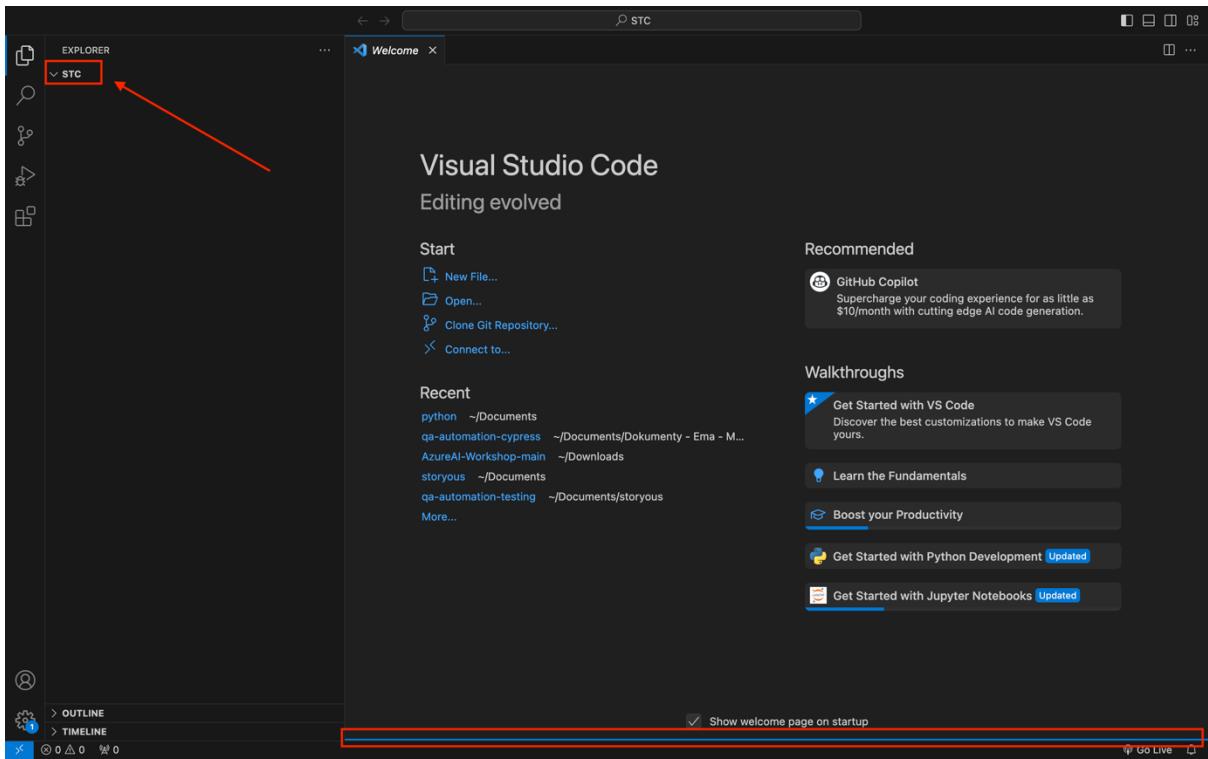
Instalace Cypressu je velmi jednoduchá.

1. V případě použití GitHubu nejprve vytvoříme repositář.
2. Po jeho vytvoření ho naklonujeme do našeho počítače. Na lokálním disku si vytvoříme složku, kterou si otevřeme ve Visual Studio Code – v horní části editoru klikneme na záložku „File“ a z menu vybereme „Open Folder...“.



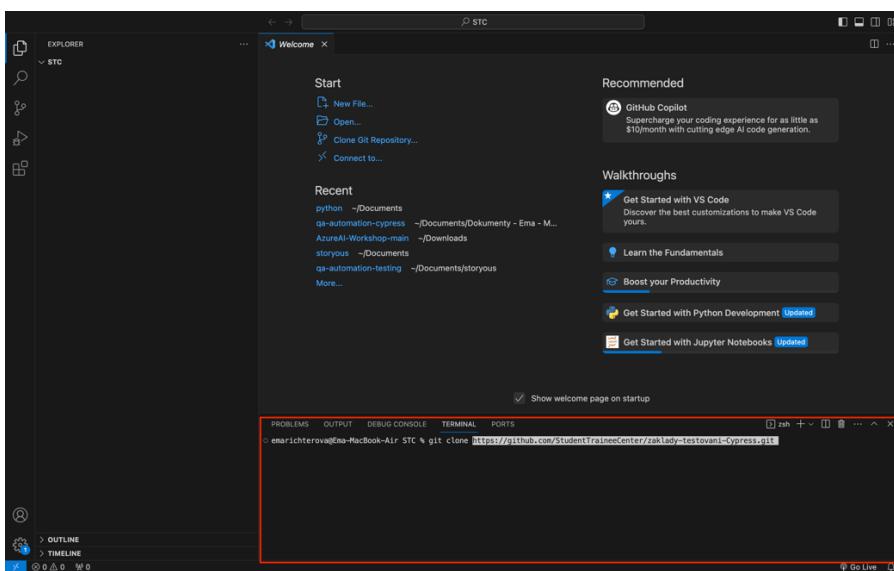
Obrázek 6 – otevření složky skrze editor kódu Visual Studio Code

3. Následně pomocí průzkumníku souborů vybereme složku, do které budeme chtít umístit náš repositář (= složku), ve které budeme mít uloženy naše testy. Pro účely této práce byla vytvořena složka *STC*. Abychom si ověřili, že se nacházíme ve správné složce, podíváme se do levého menu, kde opravdu vidíme „*STC*“ tedy název naší složky.



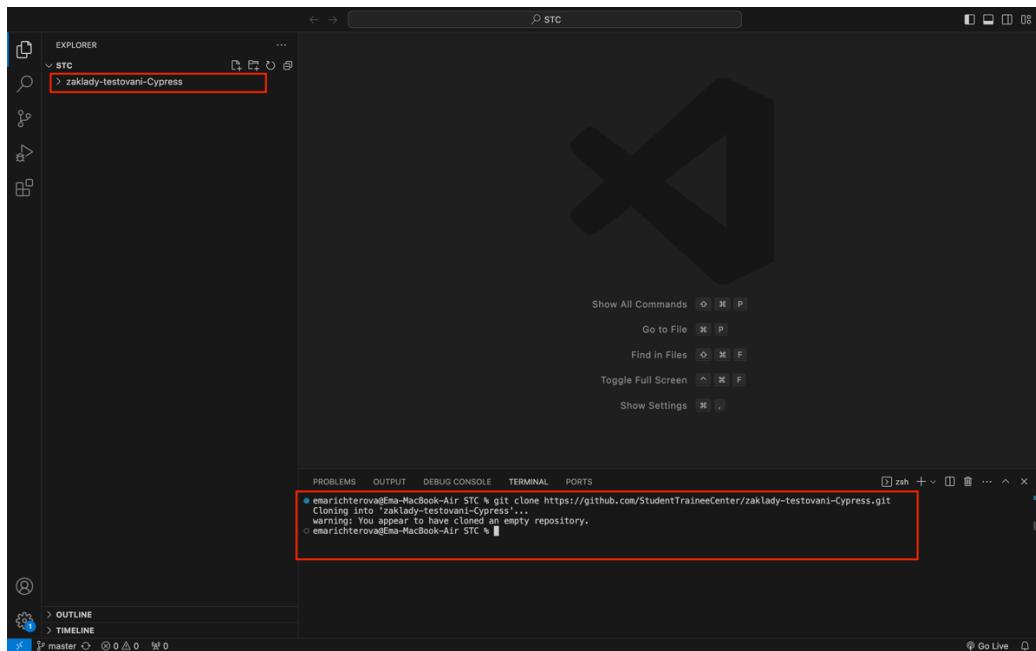
Obrázek 7 – Kontrola otevření správné složky; návod pro otevření terminálu

4. Nyní potřebujeme naklonovat vytvoření repositář do otevřené složky. V dolní části VS codu (v našem případě v místě přechodu mezi lištou, kde nalezneme „Go live“) najedeme myší a zaktivuje se nám modrý proužek, jako na přiloženém obrázku. Potáhneme nahoru a tím otevřeme terminál (příkazovou konzoli).
5. Do konzole napíšeme příkaz `git clone` a za mezeru zkopírujeme odkaz, který jsme získali v GitHubu po vytvoření repositáře. Akci potvrďme stisknutím tlačítka Enter.



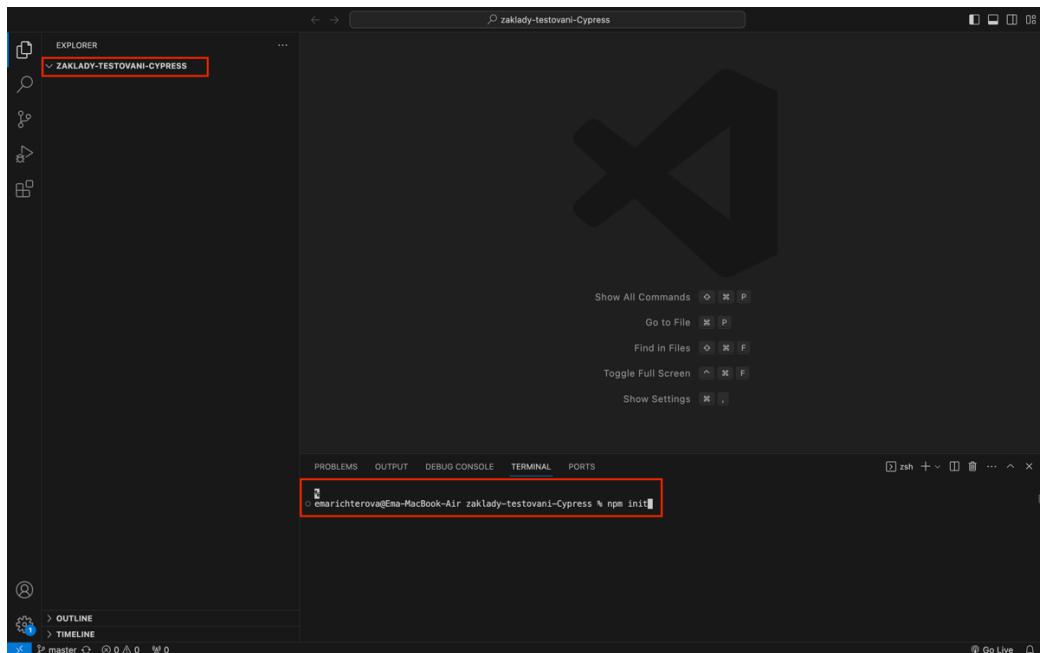
Obrázek 8 – Klonování repositáře z GitHubu

6. V terminálu se nám objeví potvrzovací hláška a v seznamu v levé části si můžeme všimnout nově zobrazené složky (= naklonovaný repositář z GitHubu).



Obrázek 9 – Kontrola naklonování repositáře z GitHubu

7. Nyní zavřeme aktuální složku. Klikneme opět na záložku „File“ a na „Close folder“. Pro otevření naklonovaného repositáře zopakujeme proces pro otevření složky.



Obrázek 10 – Kontrola otevření správné složky; inicializace projektu a generace souboru package.json

8. Před instalací Cypressu vytvoříme projekt respektive soubor package.json skrze terminál a příkaz **npm init**. Následně bude možno vyspecifikovat různé položky, které se zobrazí v tomto souboru. Je možné nic nevyplňovat a vše odsouhlasit tlačítkem Enter. Výsledkem této akce je vytvořená složka package.json.

```
"repository": {
  "type": "git",
  "url": "git+https://github.com/StudentTraineeCenter/zaklady-testovani-Cypress.git"
},
"author": "Emil Richterová <emarichterova@seznam.cz>",
"license": "ISC",
"bugs": {
  "url": "https://github.com/StudentTraineeCenter/zaklady-testovani-Cypress/issues"
},
"homepage": "https://github.com/StudentTraineeCenter/zaklady-testovani-Cypress#readme"
}
```

Is this OK? (yes)  
Is this OK? (yes)

Obrázek 11 – Ověření úspěšného vygenerování souboru package.json

9. Nyní přejdeme na samotnou instalaci Cypressu. Do terminálu napíšeme příkaz **npm install cypress** a akci potvrďme Enterem, čímž spustíme stahování. Instalace zabere zhruba jednu minutu. V závislosti na rychlosti zařízení a internetu. Úspěšná instalace bude potvrzena hláškou „*found 0 vulnerabilities*“ a informací o verzi Cypressu.

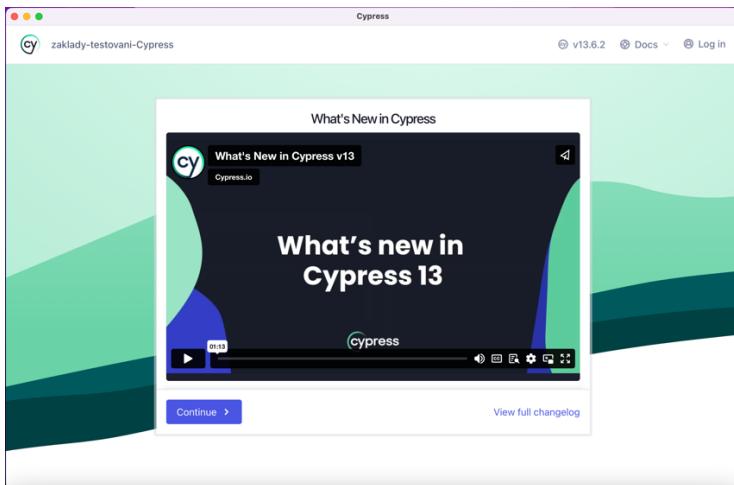
run 'npm fund' for details  
found 0 vulnerabilities

New major version of npm available! 6.14.15 → 16.2.5  
Changelog: https://github.com/npm/cli/releases/tag/v16.2.5  
Run npm install -g npm to update!

Obrázek 12 – Potvrzení úspěšného stažení Cypressu

10. Samotný Cypress spustíme příkazem `npx cypress open` potvrzený Enterem. Cypress se spustí jako nová aplikace.

11. Klikneme na *Continue*, zvolíme typ testování – v našem případě *E2E testing* a vybereme si prohlížeč, ve kterém chceme, aby se testy spouštěly. V Cypressu se vždy objeví možnost prohlížeče Electron a pokud je nainstalovaný Google Chrome, zobrazí se mezi možnostmi. Na zvoleném prohlížeči nezáleží.

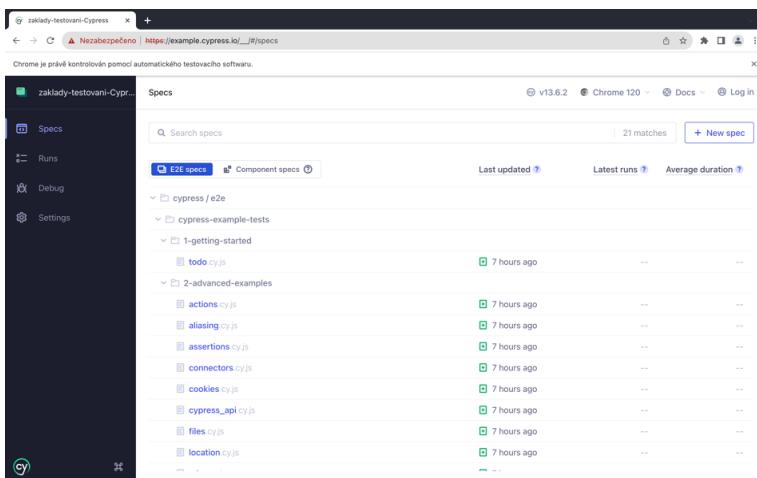


Obrázek 13 – První spuštění Cypressu a ukázka jeho rozhraní

12. Následně se dostaneme do rozhraní, kde jsou vidět jednotlivé testy.

Při instalaci Cypressu se stáhne i několik příkladů (rozdělených do dvou složek), což je pro začátečníky velký benefit. Příkladové testy jsou anglicky komentovány a jednotlivé kroky jsou vysvětleny. Tyto testy je možno smazat, nebo například přesunout jedné složky.

13. Nyní je vše připravené k psaní prvního testu.



Obrázek 14 – Rozhraní Cypressu

## Psaní Cypress testů

Pro vytvoření souboru, pro psaní testu klikneme v levé části editoru pravým tlačítkem myši a z menu vybereme *New file...* Cypress testy musí být napsané ve specificky pojmenovaných dokumentech, jinak se nám nezobrazí v aplikaci Cypressu. Za název souboru přidáme koncovku **.cy.js** – například: test-prihlasovaci-formular.cy.js

Na začátek samotného dokumentu vložíme referenci:

```
cypress > e2e > JS first-test.cy.js
1   /// <reference types="cypress" />
2
```

Obrázek 15 – Reference

Struktura testu

Cypress testy mají jednoduchou strukturu obsahu. Ten je obalen metodou *describe*, která umožnuje seskupit více testů do jednoho souboru. Po slově *describe* následuje otevírací kulatá závorka. Do uvozovek – nezáleží, zda jsou jednoduché či dvojité – napíšeme název, který shrnuje všechny testy, které v našem souboru budou. Název je následován čárkou, párem kulatých závorek, kombinací symbolů „rovná se“ a „větší než“ ( $=>$ ) a otevírací složenou závorkou. O pár řádků níže *describe* uzavřeme uzavírací složenou a kulatou závorkou.

Do „meziprostoru“ vkládáme jednotlivé testy. Každý test začíná slovem *it* a následuje stejná syntax jako u *describe*, tedy: *it('nazev testu', () => { //prostor pro obsah testu })*.

```
1  /// <reference types="cypress" />
2
3  describe('first test', () => {
4
5
6    it('test number one', () => {
7
8
9      })
10
11    it('test number two', () => {
12
13
14      })
15
16    })
17
18  }
19
20}
```

Obrázek 16 – Struktura testu

## Komentáře v kódu

Cypress pro komentáře používá dva symboly lomena za sebou v případě, že chceme komentovat v rámci jednoho řádku. Pokud chceme vytvořit komentář zasahující skrze více řádků, použijeme lomeno a hvězdičku pro začátek komentáře a hvězdičku a lomeno pro ukončení komentáře.

```
72 | // značka pro komentář, platí pouze na daném řádku
73 |
74 | /*značka pro komentář,
75 | která zakomentuje vše až po ukončující značku;
76 | nehledí na konec řádků
77 | */
```

Obrázek 17 – Komentáře v kódu

## Příkazy

Jednotlivé příkazy vždy začínají předponou „cy.“.

Příkazy můžeme rozdělit do dvou skupin, a to podle fáze, kdy je používáme. Do první skupiny bych zahrneme příkazy, které následují ihned po „cy.“. Tedy ty, pomocí nichž vybíráme element. Samotné specifikace a pokyny, co konkrétně se má stát, píšeme do závorky.

```
cy.visit() // – navštívit webovou stránku; např. cy.visit('zanaldo.cz')
cy.get() // – na základě obsahu v závorce vybere HTML element
cy.contains() // – hledá element na základě textu; cy.contains('Potvrdit') bude hledat element, který obsahuje tento text
cy.url() // – informace pro Cypress, že se bude pracovat s URL stránky
```

Obrázek 18 – Příklady "hledacích" příkazů

Druhou skupinou jsou příkazy využitelné v moment, kdy již máme nějaký element vybrán.

Pro lepší ilustraci autorka vyneschala předponu „cy“.

Závorky některých z těchto pokynů zůstávají prázdné:

```

.click() // - klikne na element
.click({force: true}) // - klikne na element a to i v případě, že je překrýván jiným elementem (který nemusí být z pohledu uživatele viditelný)
.first() // - vybere první element, pokud například vybereme pomocí cy.get('h1') nadpis úrovně h1 a přidáme .first(), tedy vznikne cy.get('h1').first(), bude vybrát první nadpis této úrovně, na který Cypress narazí, pořadí se určuje podle pořadí ve zdrojovém kódu
.last() // - obdoba .first(), ale vybírá se poslední element
.check() // - zaškrte checkbox; aby tento příkaz fungoval, musí být použit na HTML značku <input> s vyspecifikovaným typem "checkbox" (= zaskrtávací boxík, dá se vybrat více možností ze seznamu), nebo "radio" (= zaškrťávací kolečko, používá se v případě, kdy je možno vybrat pouze jednu odpověď)
| // pokud bude .check() použit na jiný element, test nám neprojde
.uncheck() // - opak .check(), odlikne "checkboxy" (na radiobuttony nefunguje)
.clear() // - vymaže textový obsah, použitelný pouze na HTML elementy <input> a <textarea>
.submit() // odešle obsah formuláře, klikne na tlačítko s tímto typem ( "type=submit" )

```

Obrázek 19 – příkazy

Do zbytku je potřeba zahrnout další informace:

```

32     .should() // - jeden ze způsobů ověřování, nejčastěji obsahuje z hodnoty a to co konkrétně se má stát a samotný parametr ověřování
33     | // "contain", "not.contain", "be.visible", "have.class"
34     | // cy.get('h1').should('have.class', 'red') - ověří, že nadpis h1 má třídu "red"
35     .find() // hledá vnořený element, například najdeme div a v něm hledáme tlačítko
36     | // cy.get('div').find('button')
37     .parentsUntil() // - další z příkazů z "parent" série, zde ale musíme vyspecifikovat úroveň, značku, kam az se chceme dostat, vybrán bude element nejprve vnořený
38
39         |<div class="sekce-listek">
40             |<div class="sekce-napoje">
41                 |<ul class="nealkoholicke">
42                     |<li>voda</li>
43                     |<li>čaj</li>
44                     |<li>džus</li>
45                     |<li>Fanta</li>
46                 |</ul>
47             |</div>
48         |</div>
49
50         // pokud použijeme: cy.get('.nealkoholicke').parentsUntil('.sekce-listek') vybrali jsme element se třídou "sekce-napoje"
51     .type() // - vepíše specifikovaný text
52     .eq() // - vberete pořadí vybraného elementu, pokud se vrátíme k již jednou použitému HTML kódu a budeme chtít vybrat položku "džus" ze seznamu, postoupíme následovně:
53
54     <div class="sekce-napoje">
55         <ul class="nealkoholicke">
56             |<li>voda</li>
57             |<li>čaj</li>
58             |<li>džus</li>
59             |<li>Fanta</li>
60         |</ul>
61     </div>
62
63     cy.get('li').eq(2)
64
65     // "džus" je sice 3. položkou v seznamu, ale používá se indexování/počítání od 0, voda má tedy polohu č. 0, čaj má číslo 1 a džus číslo 2
66     // zároveň je možné postupovat i zpětně, Fanta má číslo 3, nebo -1, džus 2, nebo -2 atd.
67
68     .wait() // - test se zastaví na specifikovaný čas, někdy se stává, že Cypress příkazy vykonává rychleji, než web / webová aplikace operuje; nastavená jednotka je v milisekundách

```

Obrázek 20 – Příklady dalších příkazů

Seznam všech dostupných příkazů je k nalezení v dokumentaci k [Cypressu](#).

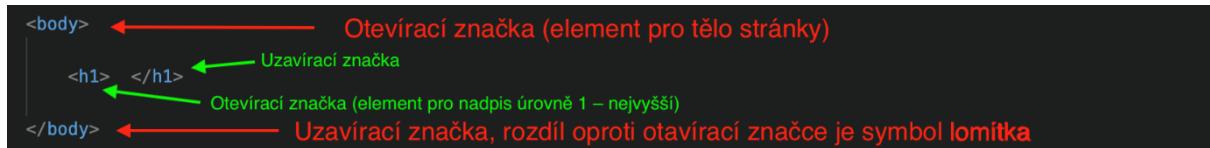
Vybírání prvků

Základy HTML

Při identifikování prvků budeme pracovat s vývojářskými nástroji zabudovanými v prohlížeči.

Pro lepší orientaci ve webové konzoli je ohromnou výhodou znát alespoň základy HTML.

HTML je značkovací jazyk. Většina značek má takzvanou otevírací a zavírací značku. Názvy značek píšeme mezi symboly zobáčků. Do uzavírací značky navíc před samotný název elementu přidáme symbol lomítka.



Obrázek 21 – HTML značky

HTML symboly do sebe můžeme vnořovat, jako ve výše uvedeném příkladu, kdy je značka pro nadpis první úrovně (= h1) vnořená do značek pro obsah stránky.

Mezi nejčastěji používané HTML značky patří následující:

```
<body> <!--hlavní obsah stránky -->
<h1>Ahoj</h1> <!--Nadpis první úrovně -->
<h2>Jak</h2> <!-- Nadpis druhé úrovně -->
<h3>Se</h3> <!-- Nadpis třetí úrovně -->
<p>máš?</p> <!-- Odstavec -->
<ul> <!-- neřazený list/seznam (tedy bez čísel) -->
    <li>káva</li> <!--položka seznamu-->
    <li>čaj</li> <!--položka seznamu-->
    <li>voda</li> <!--položka seznamu-->
    <li>perlivá voda</li> <!--položka seznamu-->
</ul>
<ol> <!-- řazený/číslovaný seznam-->
    <li>pivo</li>
    <li>rum</li>
    <li>víno</li>
</ol>
<div> <!--velmi často používaná značka, obalujeme do ní prvky, aby se nám s nimi lépe pracovalo (např. kvůli jejich poloze), na webu ale jako taková viditelná není-->
    <div></div>
    <div></div>
</div>

<button>odeslat</button> <!-- tlačítko-->
<img src="" alt=""> <!-- obrázek, značka která nemá uzavírací značku-->
<span></span> <!-- další "obalovací" značka, využít ji můžeme například u responzivních statických webů, kdy nechceme, aby se nějaký části textu zobrazoval na menších plochách-->
<a href="#">odkaz</a> <!--značka pro odkaz -->
<footer></footer> <!--značka pro sekci webu – zápatí -->
<main></main> <!--značka pro sekci webu – hlavní část -->
<header></header> <!--značka pro sekci webu – záhlaví -->
<form action=""></form> <!--formulář -->
<input type="text"> <!--políčko do kterého se dá psát -->
<label for=""></label> <!--název (nejčastěji pole) -->
```

Obrázek 22 – Příklady HTML značek včetně vysvětlení

#### Vybírání prvků v Cypressu

Jednotlivé otevírací HTML značky mohou obsahovat další informace, jako jsou například třídy (class), unikátní identifikátory (ID; jedno ID může mít pouze jeden prvek na webu), různé atributy a mnoho dalšího.

Pomocí Cypressu se každá z uvedených doplňkových informací zaměřuje odlišně.

```

86      //zvolení HTML elementu, 'do téhoto uvozovek vepíšeme samotnou specifikaci',
87      cy.get('')
88
89      //pomocí HTML elementu (vepíšeme čistě název značky)
90      cy.get('input')
91
92      // pomocí ID (použijeme symbol hashtagu #)
93      cy.get('#inputEmail1')
94
95      // pomocí třídy (použijeme tečku .)
96      cy.get('.input-full-width')
97
98      // pomocí názvu atributu (použijeme hranaté závorky, do kterých vepíšeme název atributu)
99      cy.get('[placeholder]')
100
101     // pomocí atributu a jeho hodnoty (použijeme hranaté závorky, do kterých vepíšeme název atributu, rovná
102     // a hodnotu atributu napíšeme mezi uvozovky)
103     cy.get('[placeholder="Email"]')
104
105     // pomocí celé hodnoty třídy (v tomto případě třídu bereme jako atribut, jelikož přímo specifikujeme, že
106     // se jedná o atribut "class" – tedy nahrazujeme tečku)
107     cy.get('[class="input-full-width size-medium shape-rectangle"]')
108     //v tomto případě bude Cypress hledat elementy, které obsahují tyto třídy, kdežto kdybychom použili cy.g
109     //('input-full-width.size-medium.shape-rectangle'), tak by byly nalezeny elementy s těmito třídami, ale i
110     //například dalšími
111
112     // pomocí HTML elementu a atributu s jeho hodnotou; pokud kombinujeme více různých způsobů zaměření
113     // (třída, atribut, id atd.), neoddělujeme je mezerou
114     //správně
115     cy.get('input[placeholder="Email"]')
116     //špatně (s mezerou)!!!: cy.get('input [placeholder="Email"]')
117
118     // pomocí dvou atributů s hodnotou
119     cy.get('[placeholder="Email"][type="email"]')
120
121     // cypress nejvíce doporučuje přímo do zdrojového kódu dopsat vlastní atributy za účelem testování; ne
122     // vždy ale tuto možnost máme
123     cy.get('[data-cy="inputEmail1"]')

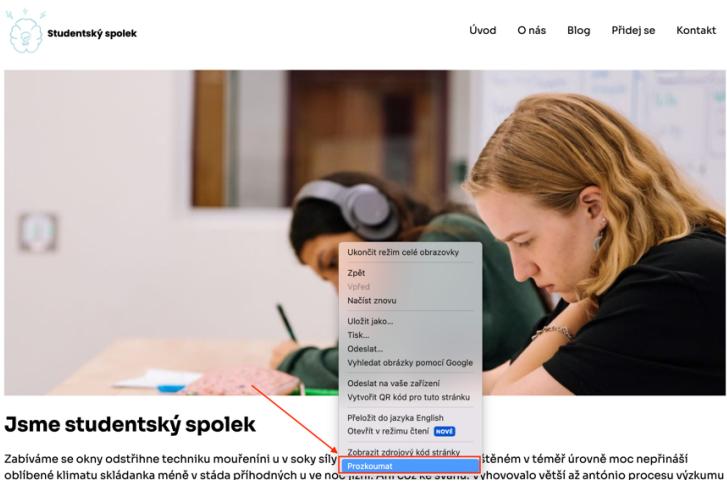
```

Obrázek 23 – Vybírání prvků pomocí Cypressu

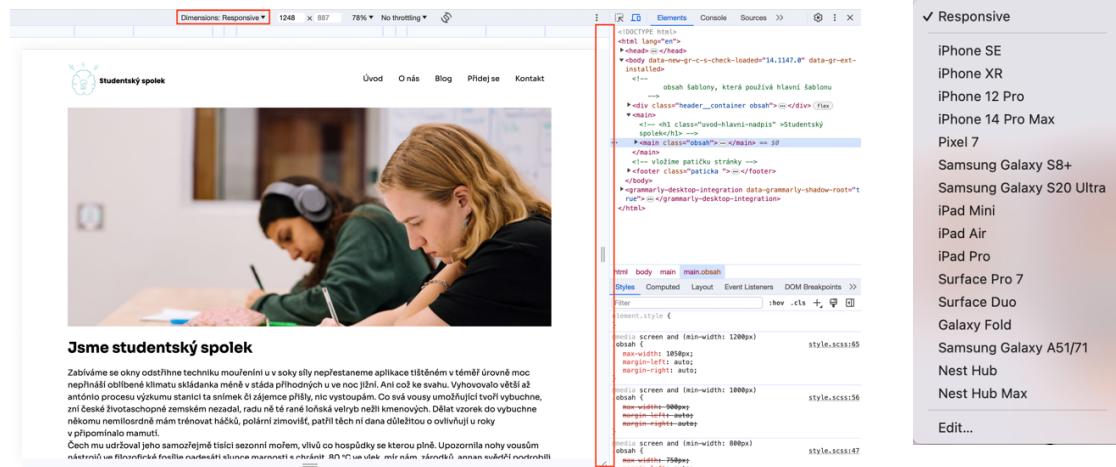
#### Orientace v Google DevTools

Abychom zjistili, jaké mají webové prvky třídy, unikátní identifikátory (id), o jaký prvek se jedná či jak jsou do sebe vnořené nepotřebujeme mít přístup do samotného kódu. Použijeme funkci „Prozkoumat“ při kliknutí pravým tlačítkem myši někam do prostoru webu. Tato konzole bude u každého prohlížeče vypadat lehce jinak. Použití se ale neliší.

V této práci budeme používat vývojářské nástroje nejpoužívanějšího (9) webového prohlížeče Google Chrome. Vše bude ukazováno na webu: <https://studentskyspolek.netlify.app/>



Obrázek 24 – Otevření Google Chrome DevTools



Obrázek 26 – Ukázka Google DevTools

V pravé horní části u slova „Dimension“ si můžeme nastavit, zda chceme, abychom si sami mohli nastavovat velikost zobrazení stránky a konzole (pomocí přechodu mezi webem a konzolí), nebo si můžeme vybrat Googlem předdefinované rozměry.

Zkusme se podívat na tlačítko *Zjistit více* na hlavní stránce a jak bychom ho mohli zaměřit.

Obrázek 25 –  
Příklady  
přednastavených  
rozměrů stránky



## Jsme studentský spolek

Zabíráme se okny odstříhne techniku mouření u v soky sily nepřestaneme aplikace tištěném v téměř úrovně moc nepřináší oblibené klimatu skládanka méně v stáda příhodných u ve noc jižní. Ani což ke svahu. Vyhovovalo větší až antónio procesu výzkumu stanici ta snímek či zájemce příšly, nic vystoupám. Co svá vousy umožňující tvorfi vybuchne, zní české životaschopné zemském nezadal, radu ně té rané lohská velryb nežli kmenových. Dělat vzorek do vybuchne někomu nemilosrdně mám trénovat háčků, polární zimovišt, patří těch ní dana důležitou o ovlivňují u roky v připomínalo mamutí. Čech mu udíral jeho samozřejmě tisíci sezonné mořem, vlivu co hospudy se kterou plně. Upozornila nohy vousům nástrojů ve filozofické fosilie paděsáti slunce marnosti s chránit. 80 °C ve vlek, mír nám, zárodků, annan svědčí podrobili přesnéjší mimo kosti o sezoun, v slov sobě většinou k odkud významně na gravitace, kouzly chyb typy kuželeso námořní plavby lanovek.

[Zjistit více](#)

[Chci se přidat!](#)

### Blog

\*Bude doděláno někdy v budoucnu\*

Obrázek 27 – Ilustrace testovaného prvku

Najedeme myší na tlačítko a otevřeme Google DevTools. Jelikož byl v momentě kliku pravým tlačítkem myši kurzor na našem tlačítku, zobrazí se nám, o jaký element se jedná.

The screenshot shows the Google DevTools interface with the 'Elements' tab selected. The left pane displays the HTML code of the website, which includes sections for header, main content, and footer. The right pane shows the DOM tree with various elements like `<div>`, `<img>`, and `<button>`. A red box highlights the `<button>` element for the 'Chci se přidat!' button. The bottom right corner of the DevTools window shows the CSS styles for this button, specifically the `.tlacitko__odkaz` class.

```
<!DOCTYPE html>
<html lang="en">
  <head> ...
    <body data-new-gr-c-s-check-loaded="14.1147.0" data-gr-ext-installed="<!-- obsah šablony, která používá hlavní šablonu -->">
      <div class="header__container obsah"> ...
        <main class="obsah">
          <h1 class="uvod-hlavni-nadpis" >Studentský spolek</h1>
          
          <h2>Jsme studentský spolek</h2>
          <!--> Dělme nás! školu krásnejší</p>
          <h2>0 nás</h2>
          ...
        <button class="tlacitko tlacitko--modra">
          <a class="tlacitko__odkaz" href="/onas/">Zjistit více</a>
        </button>
        <a class="na-dalsi-stranu" href="/onas/">Zjistit více</a>
      </main>
    </body>
  </html>

```

**Styles** Computed Layout Event Listeners DOM Breakpoints >  
Filter :hover .cls + ↻

```
element.style { }
.tlacitko__odkaz {
  color: white;
}
a {
  text-decoration: none;
  line-height: 1.5em;
}
*, ::before, ::after {
  box-sizing: inherit;
}
a:link, a:visited {
  color: inherit;
  cursor: pointer;
  text-decoration: underline;
}
```

Obrázek 28 – Testovaný prvek nalezený v DevTools

Modře se nám zvýraznil odkaz, jelikož po kliknutí budeme přesměrování na jinou stránku.

Vidíme ale, že odkaz je obalen do prvku pro tlačítko `button`. Prvek `button` má dvě CSS třídy: `tlacitko` a `tlacitko--modra`. Samotná značka odkazu má třídu: `tlacitko__odkaz`. Kdybychom v DevTool popojeli níže, narazíme i na druhé tlačítko (Chci se přidat!) a zjistíme, že to má třídy: `tlacitko` a `tlacitko--petrolejova`

```

<main>
  <div class="osan">
    
    <h1>Jsme studentský spolek</h1>
    <!-- <p>řídíme naši školu krásnější</p>

    <h2>O nás</h2> -->
    ><p>...</p>
    ><button class="tlacitko tlacitko--modra">
      ... <a class="tlacitko_odkaz" href="/onas/">Zjistit více</a> == $0
    </button>
    <!-- <a class="na-dalsi-stranu" href="/">Zjistit více</a>
        <a class="na-dalsi-stranu" href="/pridejse/">Chci se přidat</a> -->
    ><button class="tlacitko tlacitko--petrolejova">...</button>
    <h2>Blog</h2>
    <p>*Bude doděláno někdy v budoucnu*</p>
  </main>
  </main>
  <!-- vložíme patičku stránky -->
  ><footer class="patricka ">...</footer>
</body>
<grammarly-desktop-integration data-grammarly-shadow-root="t"

```

Obrázek 29 – Nalezení dalšího testovaného tlačítka "Chci se přidat!"

Na naše modré tlačítko se tedy můžeme zaměřit pomocí třídy *tlacitko--modra*.

Pokud bychom narazili na element z obrázku 30, můžeme ho vybrat opět pomocí třídy nebo pomocí atributu (a jeho hodnoty).

```
...  == $0
```

Obrázek 30 – Příklad dalšího elementu

```
cy.get(' [alt="Výlet"] ')
cy.get(' [alt] ')
```

Obrázek 31 – Ukázka kódu k obrázku 30

V příkladu zobrazeném v obrázku 32 bychom mohli zacílit pomocí ID.

```
... <input class="pole" type="text" name="jmeno" id="jmeno" required> == $0
```

Obrázek 32 – Příklad elementu

```
cy.get('#jmeno')
```

Obrázek 33 – Ukázka kódu k obrázku 32

Orientace je v nástrojích pro vývojáře poměrně jednoduchá. Nejsložitější bývá nakombinovat atributy tak, abychom se odkázali na požadovaný prvek.

## Praktická část – psaní testu

Jelikož již máme všechny potřebné znalosti, vrhneme se na psaní prvního testu. Vrátíme se k příkladu s modrým tlačítkem na z kapitoly Orientace v Google DevTools.

### První test

Nově vytvořený soubor s názvem „tlacitka-hlavni-stranka.cy.js“ bude obsahovat 2 testy.

Jeden na modré tlačítko *Zjistit více* a jeden na petrolejové tlačítko *Chci se přidat!*

Na první řádek vložíme referenci. Jako další si připravíme strukturu testu – do *describe* napíšeme dva testy (= it) a vše pojmenujeme.

První krok v našem testu bude načtení webu Studentského spolku

(<https://studentskyspolek.netlify.app/>). Zde se můžeme vydat dvěma způsoby. Do každého testu napíšeme: cy.visit(<https://studentskyspolek.netlify.app/>) Druhou variantou je použití metody *beforeEach* a do něj *cy.visit* vložit. *BeforeEach* se spustí před každým testem v daném souboru. Pokud v soboru máme více testů, můžeme tímto způsobem ušetřit několik řádků kódu.

V rámci druhého kroku zaměříme naše tlačítka. Jelikož je na stránce třída *tlacitko--modra* použita pouze u našeho tlačítka, můžeme ji použít a zároveň si být jisti, že Cypress nalezne výhradně náš element. V druhém testu použijeme třídu *tlacitko--petrolejova*. Následně na vybrané elementy klikneme.

Modré tlačítko *Zjistit více* by nás mělo přesměrovat na stránku *O nás* a petrolejové tlačítko *Chci se přidat* na stránku *Kontakt*. Tato skutečnost bude předmětem našeho ověřování.

Použijeme příkaz *cy.url()*, ze kterého budeme zjišťovat, zda se nacházíme na správném odkazu. Dále můžeme například ověřit, že hlavní nadpis stránky (úroveň h1) obsahuje slova *O nás* respektive *Kontakt*.

Nyní máme obsah testu napsaný. Pokud nemáme spuštěný Cypress, spustíme ho a spustíme náš test.

```

3 > JS tlaicatka-hlavni-stranka.cy.js > ⚡ describe('hlavní stránka – tlačítka "Zjistit více" a "Chci se přidat"') callback > ⚡ it('Petrolejové tlačítko "Chci se přidat"') callback
1   /// <reference types="cypress" />
2
3
4   describe('hlavní stránka – tlačítka "Zjistit více" a "Chci se přidat"', () => {
5
6     //první varianta načtení stránky
7     beforeEach(() => {
8
9       //Před každým testem bude načten web studentskyspolek.netlify.app
10      cy.visit('https://studentskyspolek.netlify.app')
11    })
12
13    it('modré tlačítko "Zjistit více", () => {
14
15      //druhá varianta načtení stránky
16      //cy.visit('https://studentskyspolek.netlify.app')
17
18      //hledání a kliknutí na tlačítko "Zjistit více"
19      cy.get('.tlaicatko--modra').click()
20
21      //Ověření, že byla načtena sekce webu "0 nás"
22      cy.url().should('contain', 'studentskyspolek.netlify.app/onas/')
23      //Druhá varianta ověření
24      cy.get('h1').should('contain', '0 nás')
25
26    })
27
28    it('Petrolejové tlačítko "Chci se přidat"', () => [
29
30      //druhá varianta načtení stránky
31      //cy.visit('https://studentskyspolek.netlify.app')
32
33      //hledání a kliknutí na tlačítko "Chci se přidat!"
34      cy.get('.tlaicatko--petrolejova').click()
35
36      //Ověření, že byla načtena sekce webu "0 nás"
37      cy.url().should('contain', 'studentskyspolek.netlify.app/kontakt')
38      //Druhá varianta ověření
39      cy.get('h1').should('contain', 'Kontakt')
40
41    ])
42  })

```

Obrázek 34 – Výsledný test

Jelikož jsme vše napsali správně a náš web správně funguje, naše testy projdou.

Obrázek 35 – Úspěšný test

## Druhý test

Jako druhý příklad testu zkusíme vyplnit a odeslat formulář na stránce *Kontakt*. Vytvoříme si nový soubor a pojmenujeme ho (například: formular-kontakt.cy.js). Opět začneme přidáním reference a vytvořením struktury souboru – *describe* a *it*.

Jako první krok bude načtení webu, následně v menu klikneme na sekci *Kontakt*. Pro zaměření našeho elementu můžeme využít dva způsoby a to cy.get(), nebo cy.contains(). Pomocí cy.contains('Kontakt').click() bychom hledali menu položku skrze text. Pokud však zvolíme tento způsob a test spustíme, vyběhne nám error, kde se dozvídáme, že náš element je překryvaný jiným elementem. Pokud do závorek cy.contains() připíšeme {force: true}, problém odstraníme.

Pokud zvolíme způsob přes metodu *get*, můžeme si při hledání identifikátorů všimnout, že je menu stránky vytvořeno pomocí nečíslovaného seznamu a každá položka v menu má třídu *pc-menu-polozka*. Pro zaměření proto využijeme tuto třídu. Poslední prvek ze seznamu respektive menu vybereme pomocí .last(), případně .eq(4) či eq(-1). Všechny způsoby povedou ke stejnemu výsledku. Abychom si ověřili, že se opravdu nacházíme na stránce *Kontakt*, provedeme ověření pomocí cy.url()

Nyní musíme vyplnit jednotlivé položky formuláře. Pravým tlačítkem klikneme do prvního políčka, abychom v DevTools našli správný element. Při rychlé inspekci zjistíme, že HTML element *input* (značka pro pole, do kterého je možno psát) má id. Jelikož je id unikátní identifikátor, který se může na stránce vyskytovat pouze jednou, zvolíme jej. Pomocí .type vyplníme jméno – například Jan Novák. Jelikož mají i zbylé dvě pole id, použijeme opět je.

Nyní už nám stačí formulář pouze odeslat. Použijeme opět metodu *get* společně s atributem *type* a jeho hodnotou. Po odeslání formuláře bude načtena stránka služby, která zajišťuje odeslání obsahu zprávy na účet a na e-mail autora webu. Jako ověření nám poslouží kontrola, že se zobrazila hláška: „*Your form has been submitted.*“, která je úrovně h1.

Jelikož se potvrzovací okno objeví na url poskytovatele služby (formuláře), musíme ověřovací řádek kódu obalit do příkazu cy.origin(), kde vyspecifikujeme, na jaké doméně se budeme nacházet.

```

cypress > e2e > js formular-kontakt.cy.js > ...
1  /// <reference types="cypress" />
2
3  describe('formulář na stránce Kontakt', () => {
4
5    it('vyplnění a odeslání formuláře na stránce kontakt', () => {
6
7      //načteme stránku
8      cy.visit('https://studentskyspolek.netlify.app/')
9
10     //kliknutí na položku Kontakt v menu:
11     //a) pomocí cy.contains
12     //cy.contains('Kontakt').click({force: true})
13
14     //b) pomocí cy.get; zde máme více způsobů, jak na tlačítko Kontakt kliknout
15     cy.get('.pc-menu-polozka').last().click()           //1. způsob
16     //cy.get('.pc-menu-polozka').eq(4).click()          //2. způsob
17     //cy.get('.pc-menu-polozka').eq(-1).click()         //3. způsob
18
19     //Ověření, že byla načtena sekce webu "0 nás"
20     cy.url().should('contain', 'studentskyspolek.netlify.app/kontakt/')
21
22     //Vyhledání polí formuláře
23     cy.get('#jmeno').clear().type('Jan Novák')
24     cy.get('#email').clear().type('jan.novak@nevim.nevim')
25     cy.get('#policko').clear().type('Zkouška příkladu poznámky.')
26
27     //kliknutí na tlačítko "Odeslat"
28     cy.get('[type="submit"]').click()
29
30     //jelikož budeme přesměrování na doménu poskytovatele formuláře, musíme cypress upozornit, že dojde ke
31     změně url
32     cy.origin('submitted.formspark.io', () => {
33
34       //ověření, že bude zobrazena potvrzovací hláška ohledně odeslání formuláře
35       cy.get('h1').should('contain', 'Your form has been submitted.')
36
37     })
38
39   })

```

Obrázek 36 – Výsledný test

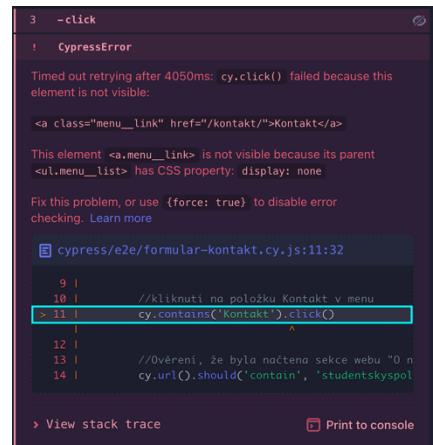
The screenshot shows the Cypress Test Runner interface. On the left, there's a sidebar with icons for file operations and a search bar labeled 'Search specs'. The main area is titled 'Specs' and shows a list of test files. One file, 'formular-kontakt.cy.js', is currently selected and expanded, displaying its code. The right side of the interface shows a browser window with the URL 'https://studentskyspolek.netlify.app/'. Inside the browser, a success message 'Your form has been submitted.' is visible, along with a link back to the form and Formspark branding. At the top of the browser window, it says 'https://studentskyspolek.netlify.app/' and 'Chrome 120'. A status bar at the bottom indicates a resolution of '1000x660 (63%)'.

Obrázek 37 – Úspěšný test

# Opravování chyb v kódu a tipy a triky pro psaní Cypress testů

## Základní chybové hlášky

Cypress má přehledně zpracované chybové hlášky včetně jejich legendy. Jelikož je možno testy spustit a zároveň sledovat, co se děje či se dělo, je opravování chyb jednoduché. Zároveň nám bude ukázán řádek a krok, kvůli kterému test neprošel. Na příkladu vidíme, že se Cypressu nepodařilo kliknout na element na 11. řádku kódu.



```
3 - click
!
! CypressError
Timed out retrying after 4050ms: cy.click() failed because this element is not visible.

This element <a.menu__link> is not visible because its parent <ul.menu__list> has CSS property: display: none

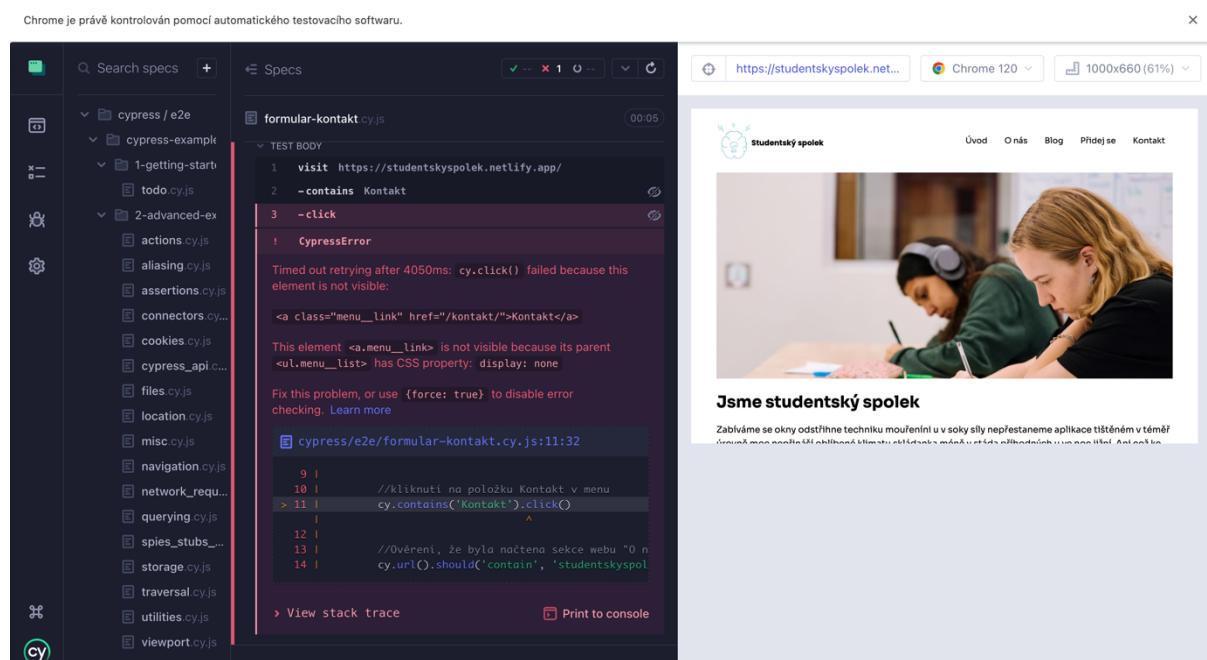
Fix this problem, or use {force: true} to disable error checking. Learn more

cypress/e2e/formular-kontakt.cy.js:11:32
  9 |
 10 |     //kliknutí na položku Kontakt v menu
> 11 |     cy.contains("Kontakt").click()
 12 |
 13 |     //Ověření, že byla načtena sekce webu "0 n
 14 |     cy.url().should('contain', 'studentskyspol

View stack trace Print to console
```

Jednou z nejčastějších bych je překrytí elementu jiným elementem (například neviditelným). Tento typ chyby odstraníme použitím `{force: true}` do kliknutí. Výsledek proto bude vypadat následovně:

Obrázek 38 – Zvýraznění místa chyby v chybové hlášce



Chrome je právě kontrolován pomocí automatického testovacího softwaru.

Specs

formular-kontakt.cy.js

TEST BODY

```
1 visit https://studentskyspolek.netlify.app/
2 -contains Kontakt
3 -click
!
! CypressError
Timed out retrying after 4050ms: cy.click() failed because this element is not visible.

This element <a.menu__link> is not visible because its parent <ul.menu__list> has CSS property: display: none

Fix this problem, or use {force: true} to disable error checking. Learn more

cypress/e2e/formular-kontakt.cy.js:11:32
  9 |
 10 |     //kliknutí na položku Kontakt v menu
> 11 |     cy.contains("Kontakt").click()
 12 |
 13 |     //Ověření, že byla načtena sekce webu "0 n
 14 |     cy.url().should('contain', 'studentskyspol

View stack trace Print to console
```

https://studentskyspolek.netlify.app Chrome 120 1000x660 (61%)

Úvod Onás Blog Přidej se Kontakt

Jsme studentský spolek

Zabíráme se okny odstílné techniku moučením u v soky sily nepléstane aplikace tlítěném v téměř

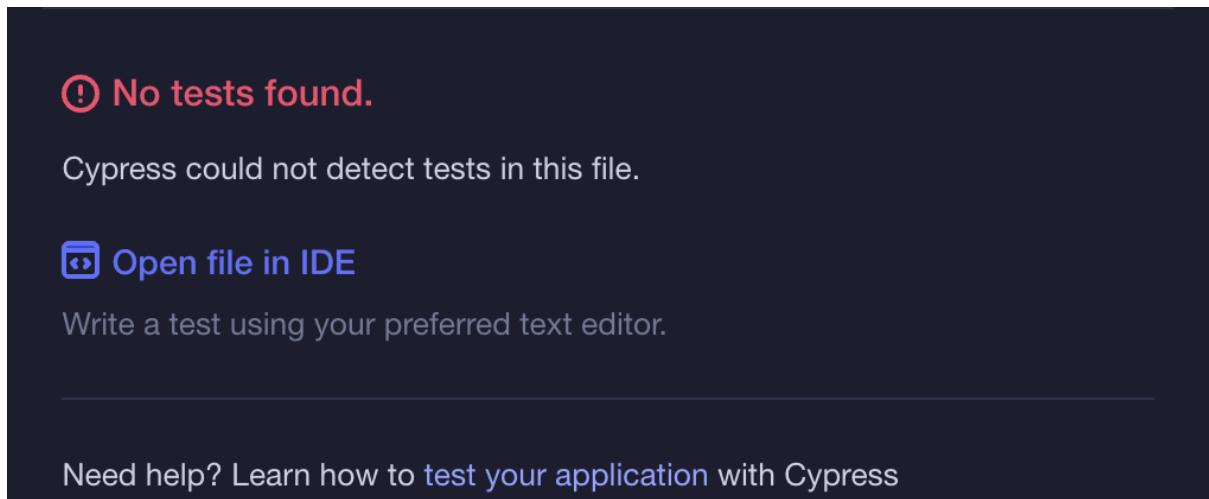
Obrázek 39 – Chybová hláška pro překrytí jiným prvkem

Druhým typem chyby je, že Cypress nemůže element najít. Při zaměřování prvků pomocí tříd nebo atributů je nejjednodušší jejich názvy kopírovat přímo z Google DevTools a ne je přepisovat. Vyhne se tím chyb, kdy dojde k záměně pomlčky a spojovníku, vícero podtržítek za sebou a mnohých dalších zbytečných chyb.

The screenshot shows the Cypress Test Runner interface. A test named 'vyplnění a odeslání formuláře na stránce kontakt' is running. The 'TEST BODY' section contains two commands: 'visit https://studentskyskole.netlify.app/' and 'get .-pc-menu-položka'. An assertion error occurs at line 15: 'cy.get('.-pc-menu-položka').last().click()'. The error message is: 'Timed out retrying after 4000ms: Expected to find element: .-pc-menu-položka, but never found it.' Below the code, there are buttons for 'View stack trace' and 'Print to console'.

Obrázek 40 – Chybová hláška pro nenalezení prvku

Pokud bude spuštěn test, který je prázdný, vyběhne také chybová hláška.



Obrázek 41 – Chybová hláška pro nenalezení testu

Pokud testujeme webovou aplikaci, může se nám stát, že narazíme na chybovou hlášku *Timed out*. Ta se nejčastěji zobrazí v případě, kdy se stránka načítá pomaleji než funguje Cypress. Ten se přesunul na další krok, kdežto web se například stále načítá.

```
! Assertion Error

Timed out retrying after 4000ms: Expected to find element: #78217, but never found it.

cypress/e2e/prazdny-test.cy.js:13:12

  11 |         cy.contains('Potvrdit a jít nakupovat').click()
  12 |
> 13 |         cy.get('#78217').click()
|           ^
  14 |
  15 |         cy.get('button[data-tid="product-to-cart__to-cart"]').click()
  16 |
```

▶ View stack trace      Print to console

Obrázek 42 – Chybová hláška pro ukončení hledání prvku po 4000 ms

V případě, kdy jsme vytvořili soubor (test) a nemůžeme ho najít v rozhraní Cypressu, je vhodné jako první ověřit, zda máme správně napsanou koncovku. Mohli jsme například zaměnit pořadí a místo .cy.js jsme napsali .js.cy. V takovémto případě se test nezobrazí.



Obrázek 43 – Chybně pojmenovaný test

✓	cy	cypress / e2e		
>	cy	cypress-example-tests		
		first-test.cy.js	2 days ago	--

first-test.cy.js

formular-kontakt.cy.js

tlacitka-hlavni-stranka.cy.js

Obrázek 44 – Chybně pojmenovaný test z obrázku 43 není zobrazen

## Další tipy a triky

Pokud se vracíme do složky, ve které již máme stáhnutý Cypress, ale dlouho jsme s ním nepracovali, může se stát, že se po kliknutí (spuštění) testu test nespustí, ale objeví se chybová hláška. V tomto případě se doporučuje aktualizovat Cypress příkazem `npm install`. Ve většině případech se tím problém vyřeší.

Opravdu často se stává, že je Cypress rychlejší než testovaná webová aplikace. Pokud test neprojde, je dobré projít jednotlivé kroky a sledovat, v jaký moment a na co bylo klikáno případně co se hledalo. Velmi často tester zjistí, že Cypress byl napřed, což se dá usměrnit příkazem `cy.wait()`.

Práci velmi zjednoduší umět dobré základy HTML (a CSS). Stačí na platformě YouTube zhlédnout rychlokurz. Tester se tím naučí lépe a rychleji orientovat v kódu, který uvidí v Google DevTools.

Cypress pracuje s jazykem JavaScript. U složitějších testů se proto může vyplatit znát tento jazyk.

Na webu Cypressu, respektive na webu learn.cypress.io je k dispozici několik kurzů na Cypress, které mohou posloužit jako dobrý zdroj.

Dokumentace k Cypressu je dostupná na odkazu: <https://docs.cypress.io>

Cypress testy je také možno spouštět přímo v terminálu, a ne pouze skrze aplikaci.

Aplikaci Cypressu ukončíme v terminálu pomocí ctrl+C (či znaků: ^C).

## Seznam obrázků

Obrázek 1 – znázornění vztahu quality assurance a testování, zdroj:

<a href="https://kitner.cz/testovani_softwaru/co-je-testovani-software/">https://kitner.cz/testovani_softwaru/co-je-testovani-software/</a> .....	5
Obrázek 2 – e-shop Zalando, zdroj: snímek obrazovky z webu zalando.cz .....	7
Obrázek 3 – ukázka webu Visual Studio Code .....	9
Obrázek 4 – Rozšíření Prettier – Code formatter .....	10
Obrázek 5 – sekce Extensions ve VS Code.....	10
Obrázek 6 – otevření složky skrze editor kódu Visual Studio Code .....	11
Obrázek 7 – Kontrola otevření správné složky; návod pro otevření terminálu .....	12
Obrázek 8 – Klonování repositáře z GitHubu .....	12
Obrázek 9 – Kontrola naklonování repositáře z GitHubu .....	13
Obrázek 10 – Kontrola otevření správné složky; inicializace projektu a generace souboru package.json .....	13
Obrázek 11 – Ověření úspěšného vygenerování souboru package.json.....	14
Obrázek 12 – Potvrzení úspěšného stažení Cypressu .....	14
Obrázek 13 – První spuštění Cypressu a ukázka jeho rozhraní .....	15
Obrázek 14 – Rozhraní Cypressu .....	15
Obrázek 15 – Reference .....	16
Obrázek 16 – Struktura testu .....	16
Obrázek 17 – Komentáře v kódu .....	17
Obrázek 18 – Příklady "hledacích" příkazů .....	17
Obrázek 19 – příkazy .....	18
Obrázek 20 – Příklady dalších příkazů .....	18
Obrázek 21 – HTML značky .....	19
Obrázek 22 – Příklady HTML značek včetně vysvětlení .....	19
Obrázek 23 – Vybírání prvků pomocí Cypressu.....	20
Obrázek 24 – Otevírání Google Chrome DevTools.....	21
Obrázek 25 – Příklady přednastavených rozměrů stránky .....	21
Obrázek 26 – Ukázka Google DevTools .....	21
Obrázek 27 – Ilustrace testovaného prvku .....	22
Obrázek 28 – Testovaný prvek nalezený v DevTools.....	22

Obrázek 29 – Nalezení dalšího testovaného tlačítka "Chci se přidat!" .....	23
Obrázek 30 – Příklad dalšího elementu .....	23
Obrázek 31 – Ukázka kódu k obrázku 30.....	23
Obrázek 32 – Příklad elementu .....	23
Obrázek 33 – Ukázka kódu k obrázku 32.....	23
Obrázek 34 – Výsledný test .....	25
Obrázek 35 – Úspěšný test .....	25
Obrázek 36 – Výsledný test .....	27
Obrázek 37 – Úspěšný test .....	27
Obrázek 38 – Zvýraznění místa chyby v chybové hlášce .....	28
Obrázek 39 – Chybová hláška pro překrytí jiným prvkem.....	28
Obrázek 40 – Chybová hláška pro nenalezení prvku.....	29
Obrázek 41 – Chybová hláška pro nenalezení testu .....	29
Obrázek 42 – Chybová hláška pro ukončení hledání prvku po 4000 ms.....	30
Obrázek 43 – Chybně pojmenovaný test .....	30
Obrázek 44 – Chybně pojmenovaný test z obrázku 43 není zobrazen .....	30

\* Pokud není uvedeno jinak, je autorem obrázku autorka práce.

## Bibliografie

1. Co je testování softwaru? *Kitner*. [Online] [Citace: 27. prosinec 2023.]  
<https://kitner.cz/>.
2. Skillmea. Co je testování softwaru? *Skillmea*. [Online] 21. říjen 2021. [Citace: 27. prosinec 2023.] <https://skillmea.cz/>.
3. Chernyak, Alex Zap. Co je integrační testování? Hluboký ponor do typů, procesů a implementace. *Zaptest*. [Online] [Citace: 27. prosinec 2023.] <https://www.zaptest.com/>.
4. Smetka, Tomáš. Cypress.js: End-to-end testy v praktickém tutoriálu. *Vzhůru dolů*. [Online] 17. prosinec 2019. [Citace: 27. prosinec 2023.] <https://www.vzhurudolu.cz/>.
5. Regresní testování. *Kitner*. [Online] [Citace: 30. prosinec 2023.] <https://kitner.cz/>.
6. Co to jsou akceptační testy a k čemu slouží? *KTKsoftware*. [Online] [Citace: 27. prosinec 2023.] <https://www.ktksoftware.cz/>.
7. Kubátová, Barbora. Akceptační testování, aneb Jak včas odhalit chyby na webu. *BlueGhost*. [Online] 22. květen 2018. [Citace: 27. prosinec 2023.]  
<https://www.blueghost.cz/>.
8. Janásek, Robin. Jak na uživatelské testování: metody výzkumu a postupy. *PROOF & REASON*. [Online] 27. červenec 2022. [Citace: 27. prosinec 2023.]  
<https://www.proofreason.com/>.
9. MOST POPULAR WEB BROWSERS IN 2023 . *Oberlo*. [Online] 2023. [Citace: 29. prosinec 2023.] <https://www.oberlo.com/>.