Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Automatizované testování webové aplikace Recruitis v nástroji Cypress

Bakalářská práce

Autor: Tadeáš Jirsa
Studijní obor: ai3-p

Vedoucí práce: Ing. Martina Husáková, Ph.D.

Hradec Králové Duben 2024

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 19.4.2024 Tadeáš Jirsa

Poděkování:

Děkuji vedoucí bakalářské práce Ing. Martině Husákové, Ph.D. za metodické vedení práce, za její volný čas, který mi věnovala, a připomínky, které v průběhu psaní této práce vznesla. Dále chci poděkovat Josefu Drábkovi, který mi věnoval svůj čas a poskytl mi cenné informace ohledně systému Recruitis.io, a zároveň Ing. Štěpánu Bartyzalovi, který mi nabídl a umožnil toto téma zpracovat.

Abstrakt

Cílem této bakalářské práce je seznámit čtenáře s testováním webových aplikací a automatizací tohoto procesu. Věnuje se konkrétněji testování webové aplikace Recruitis.io, vyvíjené společností Recruitis s.r.o, a nástroji Cypress, ve kterém tyto testy probíhají.

Teoretická část je věnovaná samotnému popisu metod a způsobů, které se při testování využívají. Dále je představen konkrétně testovací nástroj Cypress, určený k testování webových aplikací, a jsou zmíněny také jeho alternativy.

V praktické části je představena webová aplikace Recruitis.io. Dále je definován testovací scénář, podle kterého se následně postupuje při samotném implementování testů. Testy vytvořené pro aplikaci Recruitis.io jsou poté spuštěny a je zaznamenán jejich výsledek. V poslední sekci jsou popsány chyby, které psaní testů provázely.

Klíčová slova: Webová aplikace, Automatizované testování, Tester, Front-end

Abstract

Title: Automated testing of Recruitis web application in Cypress

The aim of this bachelor thesis is to introduce the reader to web application testing and the automation of this process. It focuses more specifically on testing the Recruitis.io web application, developed by Recruitis s.r.o., and the Cypress tool in which these tests are performed.

The theoretical part is devoted to the description of the methods and techniques used in testing. Furthermore, the Cypress testing tool, designed for testing web applications, is introduced and its alternatives are also mentioned.

In the practical part, the web application Recruitis.io is introduced. Furthermore, a test scenario is defined, which is then followed for the actual implementation of the tests. The tests, created for the Recruitis.io application, are then run and their result is recorded. The last section describes the errors that accompanied the writing of the tests.

Key words: Web application, Automated testing, Tester, Front-end

Obsah

1 Úvod

V dnešní době se stále větší část populace spoléhá na internet a na webové aplikace pro usnadnění běžného, ale také pracovního života. S vyššími požadavky roste také důležitost zajistit kvalitu těchto aplikací prostřednictvím efektivního testování. Jedním z nejvíce používaných přístupů k testování webových aplikací je automatizované testování, které umožňuje opakované a systematické ověřování funkcionality a uživatelského rozhraní aplikace. Kvalitně napsaným a správně využitým automatizovaným testem se oproti manuálnímu testování šetří čas, peníze a prostředky. Tato bakalářská práce se věnuje právě automatizovanému testování, konkrétně pak na automatizované testování konkrétní webové aplikace s využitím jednoho z testovacích nástrojů.

V první části se práce zabývá rozdělením testů podle způsobu použití a podle doby využití v průběhu vývoje aplikace. V další části jsou vysvětleny metody, které se obecně k testování používají, jako jsou například testy bílé a černé skříňky. Následně je detailně popsán testovací nástroj Cypress, hojně využívaný k automatizaci testovacího procesu. Poté jsou také popsány vybrané testovací alternativy nástroje Cypress a nějaké jejich výhody a nevýhody.

V druhé části je popsána webová aplikace Recruitis.io a následně se rozsáhleji věnuje implementaci získaných poznatků testování pro tuto aplikaci. V prvním kroku je definována struktura testování a testovací scénář, který je nutnou součástí pro efektivní integraci testovacího procesu do procesu vývoje aplikace. Dále je na řadě samotné psaní testů a jejich součástí a na závěr jsou popsány chyby, které byly v průběhu psaní objeveny.

2 Cíl a metodika práce

Cílem této bakalářské práce je poskytnout čtenářům přehled o možnostech a způsobech testování webových aplikací, a také o využití nástroje Cypress konkrétně k automatizovanému testování webové aplikace Recruitis.io.

Cílem druhé části této práce je vytvořit automatizované testy pro již zmíněnou webovou aplikaci.

3 Kategorie testování

Podle článku [1] není při vývoji aplikace testování jen jednorázovým procesem. Je třeba testovat průběžně, a to minimálně po každé vývojové fázi. Následující rozdělení je seřazené podle toho, v jakou chvíli od počátku vývoje se testy provádí.

3.1 Unit testy

V dokumentu [2] je uvedeno, že unit testy jsou automatizované testy, které se zaměřují na nejmenší testovatelné části aplikace, takzvané „Unity“. Podle [1] jde obvykle o testy jednotlivých komponent aplikace na úrovni modulů a tříd. Tento druh testování provádí nejčastěji už samotní vývojáři. Každý jednotkový test by měl být nezávislý na ostatních, což znamená, že by měl být schopen běžet samostatně a v jakémkoli pořadí. To umožňuje vývojářům identifikovat a opravit chyby v konkrétních částech kódu bez nutnosti procházet celý systém. Do češtiny se tyto testy překládají jako „jednotkové testy“. Tyto testy by měly být jasně definované, měly by být volané jedním příkazem a jejich výstup by měl být jednoznačný – test buď prošel, což znamená, že testovaná jednotka funguje správně, nebo neprošel, což znamená, že v testované jednotce byla nalezena chyba.

3.2 Integrační testy

Při provádění integračních testů je důležité rozlišovat mezi vnitřní a vnější integrací. Vnitřní zahrnuje vzájemnou komunikaci mezi jednotlivými částmi aplikace a u vnější se jednotlivé aplikace propojují do větších funkčních celků. Oba tyto druhy integrací vyžadují provádění integračních testů. Integrace je vývojem aplikací považována za kritickou oblast, a proto jsou integrační testy důležité. Tyto testy se obvykle provádějí postupně od jednotlivých modulů směrem k větším celkům, přičemž se nejprve testují rozhraní jednotlivých modulů, následuje spojení modulů do větších celků, které mají smysl pro testování. Nakonec se testuje kompletní aplikace. Pokud jde o vnější integraci, při které se aplikace integrují od různých výrobců, vyžaduje to často kooperaci mezi různými vývojovými týmy.

3.3 Smoke testy

Autoři publikace [3] uvádějí, že se tyto testy využívají po dokončení vývoje aplikace v momentě, kdy je již spustitelná, což zpravidla nastává ke konci integračního testování. Smoke testy jsou krátké a slouží především jako ověření, že jsou všechny části aplikace implementovány, nainstalovány a spuštěny. Zaměřují se zejména na hlavní funkce aplikace, které nejsou často měněny, a nezabývá se drobnými detaily. Smoke testy jsou často považovány za první krok v procesu testování softwaru. Pokud tyto testy procházejí, pak se obvykle předpokládá, že aplikace je dostatečně stabilní pro další testování. Pokud naopak tyto testy selžou, pak to obvykle znamená, že existují závažné problémy, které je třeba řešit před dalším testováním. Po úspěšném splnění těchto testů přichází na řadu systémové testy.

3.4 Systémové testy

V dokumentu [1] se píše, že pokud se v dnešní době mluví o testování, má se na mysli právě tento druh testů. Jedná se o test softwaru, který se zaměřuje na ověření, že celý systém funguje správně jako celek. Testuje se, že aplikace správně plní úlohu, pro kterou byla vyvinuta, že byly pokryty všechny požadavky od zákazníka, nebo že správně reaguje na různé vstupy a komunikuje s jinými systémy a komponentami. To může zahrnovat testování, jak aplikace zpracovává data, jak reaguje na chyby a jak se chová v různých operačních prostředích. Dále se testuje, že je aplikace schopna správně zvládnout nestandardní a výjimečné situace, že správně reaguje na neočekávané vstupy, že se dokáže vyrovnat s chybami a testuje také, jak se chová pod nadstandartní zátěží. Tyto testy probíhají v několika kolech. Po každém otestování aplikace se nahlásí nalezené chyby, a po opravení těchto chyb se spouštějí testy znovu a celý proces se opakuje do opravení všech nalézaných chyb.

3.5 Akceptační testy

Tento druh testů je nejdůležitější z pohledu dodání samotné aplikace zákazníkovi. Testuje, jestli aplikace splňuje všechny požadavky zákazníka. Testy můžou provádět buď ti, co již prováděli systémové testy, nebo si zákazník určí vlastní testovací tým. Zákazník si určí, za jakých podmínek je software přijatelný. Těmto podmínkám se

říká akceptační kritéria, a zahrnují výčet požadovaných funkcí, testovacích scénářů a maximální počet různých typů chyb, které může software obsahovat, aby tato kritéria splňoval. Autor publikace [3] říká, že akceptační testy jsou obvykle poslední fází testování před tím, než je software dodán zákazníkovi. Pokud software splňuje všechna akceptační kritéria, je považován za připravený k dodání.

3.6 Regresní testy

Tyto testy ověřují, že oprava chyby v aplikaci, nebo přidání nové funkcionality neovlivnily chod aplikace v místech, kde by být ovlivněn neměl. Zjišťují se odchylky mezi výstupy aplikace před a po provedení změn. Tyto odchylky mohou naznačovat, že změny v kódu měly vedlejší účinky, které mohou ovlivnit funkčnost aplikace. Tento typ testů se nejčastěji automatizuje, protože se zná běžný výstup, který se dá jednoduše porovnat s novým výstupem, a tím rychle odchylky, které důsledkem opravy nebo přidáním funkcionality nastaly, identifikovat.

4 Metody testování

Rozlišujeme několik metod testů, které se k testování aplikací používají.

4.1 Testy splněním a selháním

Někdy se uvádí pod pojmy pozitivní a negativní testy. Využívají se velmi často, hlavně v kombinaci s testy černé skříňky. Obvykle se spouští nejprve testy splněním, a pak po úspěšně splněném testu následují testy selháním. Tato metoda testování se využívá zejména v systémové části testování.

4.1.1 Testy splněním

Testy splněním neboli pozitivní testy, využívají pro vstupní data v aplikaci jen ta data, která musí aplikace vždy akceptovat. Kontroluje se, že je získaný výstup shodný s očekávaným výstupem.

4.1.2 Testy selháním

Testy selháním, také uváděné jako negativní testy, využívají jako vstup pouze nestandartní data. Na rozdíl od testů splněním, tyto testy se snaží aplikaci neočekávaně ukončit a ověřují, že výstup aplikace neobsahuje žádná nežádoucí data, tj. žádná data, která nejsou mezi očekávaným výstupem.

4.2 Funkční a nefunkční testy

4.2.1 Funkční testy

Autor článku [5] uvádí, že funkční testy zjišťují, jestli aplikace pracuje podle požadavků, a to procházením všech funkcí aplikace. Tester po zadání vhodných vstupů zkontroluje výstupy a porovná je s očekávanými výsledky. Toto testování zahrnuje kontrolu uživatelských rozhraní, rozhraní API, databází atd. Využívá jak testy splněním, tak testy selháním.

4.2.2 Nefunkční testy

Naopak nefunkční testy, podle článku [5], se zaměřují na nefunkční aspekty systému, jako je například výkonnost, spolehlivost, nebo bezpečnost. Testy se

provádí, hlavně tedy kvůli otestování výkonnosti, jako by byla aplikace vytížená při klasickém, očekávaném provozu, ale i v extrémním, nadstandardním zatížení.

4.3 Testování bílé a černé skříňky

4.3.1 Testování bílé skříňky

Jak uvádí autor článku [6], při použití této testovací metody zná tester vnitřní strukturu aplikace a svoje testování zaměřuje na implementaci a dopad kódu. Testuje tedy jednotlivé části kódu a implementované algoritmy a metody. Cílem této metody je ověřit správnost a účinnost kódu. Nevýhodou této metody je, že tester nemusí být objektivní a může testování přizpůsobit fungování kódu aplikace, kde se už může vyskytovat chyba, z toho důvodu pak nemusí být výsledek testování správný.

4.3.2 Testování černé skříňky

Článek [6] se zmiňuje také o tom, že tato metoda se zaměřuje hlavně na vnější design a vstupy a výstupy aplikace. Uživatel této metody nemá přístup ke zdrojovému kódu a nezná interní fungování aplikace, ví pouze, že má dostat určitý výstup. Cílem této metody je zjistit, že aplikace funguje podle požadavků a specifikací.

4.3.3 Testování šedé skříňky

Autoři dokumentu [23] uvádí, že, jak už název napovídá, se jedná o kombinaci testování bílé a černé skříňky. Software se testuje jako v případě černé skříňky, k tomu je ale přidáno lehké nahlédnutí do kódu softwaru jako v případě bílé skříňky. Tato kombinace umožňuje lépe pochopit příčinu problémů, které se můžou objevit. Webové stránky jsou pro tuto metodu jako dělané, díky lehce pochopitelnému a přehlednému HTML kódu i pro testery bez znalostí programování.

4.4 Statické a dynamické testování

4.4.1 Statické testování

V článku [7] se pojednává o tom, že statické testování je metoda testování, při které není potřeba mít spuštěnou testovanou aplikaci. Když tedy není testována

funkcionalita běžící aplikace, je kontrolován zdrojový kód a funkční požadavky. Tyto testy obvykle probíhají v raných fázích vývoje, protože se hledají možné zdroje selhání ještě, než se dá vůbec aplikace spustit.

4.4.2 Dynamické testování

Autor článku [8] říká, že na rozdíl od statického testování to dynamické vyžaduje interakci se spuštěnou testovanou aplikací. Hlavním cílem této metody je otestovat chování spuštěné aplikace s dynamickými proměnnými a ověřit, že se například výstupy dynamicky mění v závislosti na vstupech.

5 Automatizace testování

Obecně testování je velmi důležité pro správné dlouhodobé fungování softwaru, nebo kteréhokoliv produktu i z jiného odvětví, než jsou informační technologie. Tato bakalářská práce se věnuje oboru informačních technologií, proto zde nebudou zahrnuty jiné obory, i když by se k tématu testování vztahovaly. V moderním světě existují dva druhy testování – manuální a automatizované.

5.1 Manuální testování

Tímto pojmem se myslí testování, při kterém je vyžadována přítomnost testera a jeho interakce s testovaným softwarem. Z důvodu možného selhání lidského faktoru ale nemusí být přesné. Právě kvůli rychlosti, jakou je člověk schopen testovat, nejsou manuální testy oproti těm automatizovaným efektivní. Na druhou stranu se oproti automatizovaným testům mnohem lépe hodí k testování, při kterém je vyžadováno rozhodování a lidská intuice. K manuálnímu testování není potřeba žádných programovacích znalostí, je ale za potřebí se v prostředí softwaru mnohem lépe orientovat.

5.1.1 Výhody manuálního testování

Podle [21] jsou výhody manuálního testování následující:

- Může být nákladově efektivnější v závislosti na typu testované aplikace.
- Je přizpůsobivější a umožňuje testerům upravovat svůj přístup podle toho, jak odhalují nové problémy.
- Intuitivní a vhodné pro testování použitelnosti a přístupnosti.

5.1.2 Nevýhody manuálního testování

Podle [21] jsou nevýhody manuálního testování následující:

- Vyžaduje lidskou práci, která je časově náročnější než automatizovaný software.
- Více náchylné k chybám a produkuje méně konzistentní výsledky.

- Nižší pokrytí testů v případě testování systému s velkým počtem testovacích případů.

5.2 Podmínky pro automatizaci

Pro automatizaci testu je nezbytné, aby test splňoval přesně stanovené podmínky. Bez jejich splnění by automatizace nemusela být efektivní. Každý zadavatel může mít své vlastní specifické požadavky, zde je výčet několika z těch, které je třeba brát v úvahu:

5.2.1 Opakovatelnost

Automatizace testu má smysl pouze v případě, že tester bude ten stejný test chtít využít ještě alespoň jednou. Automatizace testu, který je pouze na jedno použití nedává smysl a je velmi neefektivní.

5.2.2 Izolovanost

Pro dosažení kvalitní automatizace testování by měla být pro každý jednotlivý test vytvořena vlastní testovací databáze a prostředí. To je velmi důležité pro fungování paralelně spuštěných testů, protože by u nich mohla nastat situace, ve které by se dva testy ovlivnily, a kvůli tomu by testy neprošly. To stejné ale platí i pro testy spuštěné sériově za sebou, po každém testu je nutné vytvořenou databázi a prostředí obnovit do stavu před spuštěním, aby každý další test využíval ta stejná, předem známá data. V případě že takový test není izolovaný, může při selhání jednoho testu dojít i k selhání testů navazujících.

5.2.3 Jednoduchá údržba

Vývoj testů může být velmi náročný proces. Po dokončení vývojové fáze testu by se v případě, že se změnila testovaná část aplikace, měl jen jednoduše upravit příslušný kód testu. Pokud údržba není možná nebo je náročná, automatizace konkrétního testu se nevyplatí. Cílem automatizace je totiž co nejméně vyžadovat zásah člověka.

5.3 Automatizované testování

Podle článku [22] existuje velké množství testů, a většina z nich může být automatizována. Jak již bylo zmíněno, pro testy použitelnosti je výhodnější využít

manuální testy, naopak například unit testy, akceptační testy, integrační testy, nebo testy výkonu se nejčastěji pro zefektivnění testování automatizují.

5.3.1 Výhody automatizovaného testování

Podle [21] jsou výhody automatizovaného testování následující:

- Obvykle rychlejší provedení (zejména při paralelním provádění testů).
- Větší pokrytí testů, protože dokáže efektivně zpracovat velký objem testovacích případů.
- Méně náchylné k lidským chybám.

5.3.2 Nevýhody automatizovaného testování

Podle [21] jsou nevýhody automatizovaného testování následující:

- Vyžaduje velkou počáteční investici času a zdrojů.
- Nemůže zachytit cenné vstupy, jako je vnímaná uživatelská přívětivost.
- Automatické nástroje jsou často složitější a náročnější na zdroje.

6 Testovací nástroj Cypress

6.1 Základní popis

Cypress je open-source framework, založený na skriptovacím jazyce JavaScript, a je jedním z mnoha moderních testovacích nástrojů k automatizovanému testování front-endu webových aplikací. Nástroj umožňuje testy napsat, spustit, i je později ladit (debugovat). Možnou nevýhodou tohoto nástroje může být, že umožňuje testování pouze webových aplikací.

6.2 Klíčové vlastnosti

Cypress je postaven na Node.js, což znamená, že může používat velké množství balíčků a modulů z celého Node.js ekosystému, a uživateli umožňuje psát testy v JavaScriptu, který je s webovou aplikací úzce spojený.

6.2.1 Automatické sledování změn

Cypress sleduje všechny změny v testovacím kódu, resp. pokud tester provede změnu, test se automaticky spustí znovu. Není tedy potřeba testy spouštět ručně pořád dokola, což umožňuje rychlejší a efektivnější testování. Na druhou stranu, pokud trvá test delší dobu a tester chce v průběhu testu provést malou změnu, je pro něj znovuspuštění testu nežádoucím chováním.

6.2.2 Jednoduché psaní testů

Testy, jak již bylo zmíněno, se píší v jazyce JavaScript, který je vývojářům webových aplikaci blízký, a proto by psaní těchto testů nemělo dělat problém. Ne však všem může vyhovovat JavaScript, jiné jazyky ale nejsou podporovány. Cypress také umožňuje přímý přístup k DOM (= struktura webové stránky), díky čemuž se dá na testované stránce velmi jednoduše a rychle vyhledávat. Nejčastěji jsou cílem vyhledávání tlačítka, nebo textová pole.

6.2.3 Snadná integrace

Cypress umožňuje integraci s různými nástroji pro automatizované testování. Nejčastěji se integrují CI (Continuous Integration) systémy, jako je například GitLab CI/CD.

6.3 Funkcionality

Podle dokumentu [4] nabízí Cypress mnoho funkcionalit, díky kterým bude práce s ním pohodlnější a rychlejší.

6.3.1 Cestování časem

Cypress v průběhu testu pořizuje snímky, jak v jednotlivých krocích samotný test vypadá. Později se k těmto snímkům lze vrátit a zkontrolovat, co přesně každý testovací příkaz udělal. Po selhání testu je také možné se vrátit zpět do daného okamžiku a zjistit, kvůli čemu test selhal.

6.3.2 Laditelnost

Tato vlastnost, přeložena z angl. debuggability, znamená, že v případě selhání testu Cypress vrátí chybovou hlášku a další informace o tom, kde přesně a proč se stala chyba. Testy lze ladit přímo v prohlížeči díky klasickým nástrojům pro vývojáře (například Windows zkratka pro prohlížeč Chrome na otevření nástrojů pro vývojáře je: Shift + CTRL + J). Na obrázku 1 je příklad chybové hlášky, která označí přímo příkaz, na kterém test selhal, a zobrazí k němu podrobnější informace.

Obrázek 1: Příklad chybové hlášky (vlastní práce autora)

6.3.3 Automatické čekání

Cypress automaticky čeká na provedení jednotlivých příkazů, proto není vhodné doplňovat testovací kód o další zbytečná čekání, což by bylo velmi neefektivní. Někdy se ale může stát, že například kvůli pomalému načítání webové stránky test selže, proto se dá výchozí maximální délka čekání na provedení jednoho příkazu nastavit ve speciálním konfiguračním souboru.

6.3.4 Snímky obrazovky a videa

Cypress test se dá spustit nejen přímo v prohlížeči a přímo sledovat průběh testu, ale dá se spustit i z příkazového řádku na pozadí. V tomto případě není vidět, jak test probíhá a nelze se tedy ani podívat na pořízené snímky, ke kterým se vrací díky cestování časem. Jak ale v případě selhání testu zjistit, v jakém místě se stala chyba? k tomu právě slouží snímky obrazovky a videa. Video zaznamenává celý průběh testu, tato funkcionalita lze ale v případě potřeby vypnout. Snímky obrazovky pak zaznamenají snímek, ve kterém test selhal.

6.3.5 Cypress Cloud

Jedná se o placenou funkcionalitu, která umožňuje testy spouštět paralelně, čímž může výrazně zkrátit dobu testování. Nabízí také novou možnost automatického zastavení probíhajícího testu v případě, že selže, čímž šetří využité prostředky. Dále také poskytuje mnohem podrobnější přehled o dřívějších testováních a díky tomu umožňuje testerovi lépe debutovat.

6.4 Typy testů

Cypress umožňuje psát více typů testů, jako jsou end-to-end testy, komponentové testy, integrační testy a jednotkové testy. Obecně lze pomocí Cypressu testovat cokoliv, co běží ve webovém prohlížeči.

6.4.1 End-to-end testy

Podle článku [9], v sekci Testing Types, Cypress vznikl a byl designován pro end-to-end testování jakéhokoliv softwaru, který běží v prohlížeči. Takový test webovou aplikaci spustí a v otevřeném webovém prohlížeči provádí úkony, které by v reálném světě prováděl běžný uživatel.

Tyto testy otestují celou webovou aplikaci od front-endu až po back-end, včetně integrací aplikace. Tímto se otestuje funkčnost aplikace jako celku. Protože se ale testuje kompletně celá aplikace, jsou tyto testy náročné na psaní i na pozdější údržbu a jejich průběh může být velmi zdlouhavý.

6.4.2 Komponentové testy

Jak je zmíněno v článku [9] v sekci Testing Types, Cypress komponentové testování umožňuje testovat jednotlivé komponenty z několika různých moderních front-endových knihoven a aktuálně podporuje některé vývojové servery a frameworky, používající knihovny React, Vue, Angular a Svelte.

Na rozdíl od end-to-end testování, tento test nenavštěvuje jednotlivé URL adresy a nenačítá jejich obsah. Místo toho mohou být jednotlivé komponenty testovány

zvlášť. Tyto testy jsou mnohem rychlejší na psaní i jejich údržbu, a dokonce i čas jejich průběhu je velmi krátký.

6.4.3 Další testy

Cypress poskytuje další typy testování mimo již dva zmíněné hlavní. Dalším možným je API testování, protože dokáže provádět libovolná http volání. Stále však přibývá nových typů testů, díky stále se rozšiřujícímu spektru pluginů.

6.4.3.1 Mailosaur

Jedná se o jeden z pluginů vytvořených Cypress komunitou. Díky němu se rozšíří možnosti testování o emailové a sms testování. Podle dokumentu [19], se dá pomocí tohoto pluginu otestovat například ověřovací email, obsah zaslaného emailu, nebo pracovat s emailovými přílohami. S tímto rozšířením se dá v prémiové verzi testovat i ověřovací sms.

6.4.3.2 Applitools

Jak autoři článku [20] uvádí, Applitools, jako jeden z pluginů umožňujících vizuální testování, přidává do Cypressu možnost zkontrolovat, jak webová stránka v konkrétním momentě vypadá. Tester ví, jak by měla stránka vypadat, proto funkcí přidanou pluginem nastaví konkrétní vzhled, a poté každé spuštění testu zkontroluje, jestli aktuální vzhled odpovídá tomu požadovanému.

6.5 Průvodce pro práci s Cypressem

Tato dílčí sekce se věnuje samotné manipulaci s Cypressem. Nejdříve je popsán proces instalace, dále je představen způsob psaní testů, demonstrovaný na jednoduchém modelovém příkladu, a na závěr je popsáno spuštění a následné ladění.

6.5.1 Instalace

Způsobů instalace má Cypress více. Podle [24] je můžeme rozdělit na dvě skupiny.

6.5.1.1 Instalace z příkazového řádku

U všech způsobů je nejprve nutné otevřít v příkazovém řádku složku projektu, ve kterém budeme chtít spouštět. Toho se dá dosáhnout pomocí příkazu v kódu 1.

```
cd /cesta/do/projektu
```

Programový kód 1: Otevření složky projektu (upraveno autorem práce dle zdroje [24])

Dále je nutné mít stažený balíček Node.js. Díky němu je pak možné zavolat příkaz *npm init*, v případě, že ještě cílová složka neobsahuje soubor *package.json*, nebo složku *node_modules*.

Jednou z možností je instalace pomocí npm příkazem níže. Tento způsob je doporučovaný samotnými vývojáři Cypressu, protože díky němu lze Cypress verzovat, stejně jako ostatní závislosti projektu, a zároveň usnadňuje spouštění Cypressu v CI.

```
npm install Cypress --save-dev
```

Programový kód 2: Instalace pomocí npm [24]

Dalšími způsoby může být instalace pomocí yarn, nebo pnpm, přičemž druhá z těchto možností vyžaduje předem připravené lokální prostředí pnpm. Lze použít i příkazy z kódu 3.

```
yarn add cypress --dev
pnpm add cypress -D
```

Programový kód 3: Instalace pomocí yarn a pnpm [24]

6.5.1.2 Přímá instalace

Cypress lze stáhnout i přímo z internetu kliknutím na odkaz https://download.cypress.io/desktop, poté stačí jen rozbalit stažený .zip soubor a spustit aplikaci. To ale není nejvhodnější způsob a slouží spíše pro vyzkoušení samotného nástroje bez použití Node.js.

6.5.2 Otevření Cypressu

Po úspěšné instalaci je možné Cypress otevřít, v případě přímé instalace jen spuštěním aplikace, v případě instalace z příkazového řádku pomocí příkazů v kódu 4 (záleží na tom, jaký příkaz byl použit k instalaci).

```
npx cypress open – pro npm
yarn cypress open – pro yarn
pnpm cypress open – pro pnpm
```

Programový kód 4: Otevření aplikace Cypress [25]

Cypress ale není nutné otevírat, pro spuštění testů je možné zadat do konzole jednoduchý příkaz z kódu 5.

```
npx cypress run
```

Programový kód 5: Spuštění testu [26]

Tento příkaz nespustí celou aplikaci, ani její uživatelské rozhraní, ale spustí test na pozadí. Průběh takového testu je možné sledovat přímo v konzoli, popřípadě pokud je zapnutá možnost nahrávání videa průběhu testu, je možné si po doběhnutí testu záznam pustit.

Po otevření je již možné spustit připravené testy vybráním prohlížeče, ve kterém chceme test spustit a následným zvolením konkrétního dílčího testu nebo sady testů. Pokud však testy připravené nejsou, musí se nejprve napsat.

6.5.3 Psaní testů

Abychom mohli spustit testy, nejprve je nutné nějaké napsat. Pro napsání takového kódu je možné využít například VS Code, dostupný na https://code.visualstudio.com. V následující části je popsán proces psaní testů. Nejprve je ale důležité stanovit si cíl, kterého by mělo testování dosáhnout, a připravit si všechny části, které bude chtít uživatel otestovat.

6.5.3.1 Důležité operace

Cypress přejal základní BDD (Behavior-Driven Development) strukturu od frameworku Mocha. Pro správné a efektivní psaní testů je dobré nejprve příkazům porozumět.

6.5.3.1.1 before, after, beforeEach, afterEach

Každý z těchto čtyř hooků, které byly převzaty z frameworku Mocha, slouží k lepšímu oddělení testů. Before funkce se provádí jednou, a to na úplném začátku celého testování. *After()* funguje stejně jako *before()* s tím rozdílem, že se provádí na úplném konci. Stejně se liší i funkce *beforeEach()* a *afterEach()*, jen se tyto bloky spouští před každým blokem *it()*, eventuálně po něm.

6.5.3.1.2 describe, it

Obě tyto funkce slouží k oddělení jednotlivých testů a byly také Cypressem převzaté od frameworku Mocha. Funkce *describe()* odděluje větší celky a může obsahovat i více funkcí *it()*, zatímco *it()* slouží k oddělení jednotlivých částí testu. V kódu 6 je příklad jejich užití, spolu s příkladem hooků z předchozího bodu.

```
before(() => {
// kód pro vykonání jednou před celým testováním
})
beforeEach(() => {
// kód pro vykonání před každým testem
})
afterEach(() => {
// kód pro vykonání po každém testu
})
after(() => {
// kód pro vykonání jednou po celém testování
})

describe("domovská stránka", () => {
  it("test menu", () => {
// testovací kód
  })
  it("test nastavení", () => {
// testovací kód
  })
})
```

Programový kód 6: Struktura testu (vlastní práce autora)

6.5.3.1.3 visit

Funkce *cy.visit()* přijímá jako parametr webovou stránku, kterou chce uživatel otestovat. Tento příkaz je nutností každého bloku *it()*, protože jím musí každý test VŽDY začít – bez navštívení webové stránky není co k otestování.

6.5.3.1.4 get

Nejzákladnější a nejčastěji používaný příkaz *cy.get()*, který přijímá jako parametr HTML element, který se po načtení požadované webové stránky snaží najít. Pokud takový element nenajde, test selže. Pokud ale naopak takový element najde, pokračuje test dál. Na tento příkaz navazují další funkce, které s nalezeným elementem dále pracují.

6.5.3.1.5 should, and

Jedná se o základní funkci, která se řetězí například za příkazem *cy.get()*, a její vstupní parametr může být jedno z mnoha tvrzení, které Cypress užívá z knihovny Chai. Pokud bude chtít uživatel například zjistit, jestli element s identifikátorem „button“ má zároveň třídu „submit-button“, použije příkaz z kódu 7.

```
cy.get(„#button“).should(„have.class“, „submit-button“)
```

Programový kód 7: Ukázka použití should (vlastní práce autora)

Pokud chce ale uživatel ověřit více tvrzení najednou, za funkci *should()* lze přidat další funkce *and()*, jak je znázorněno v kódu 8.

```
cy.get(„#button“).should(„have.class“, „submit-button“)
  .and(„be.visible“).and(„contain“, „Submit“)
```

Programový kód 8: Ukázka použití and (vlastní práce autora)

6.5.3.1.6 click

Jako jeden z mnoha akčních příkazů se řetězí za příkazy, jako může být například *cy.get()* nebo *cy.contains()*, které odkazují na konkrétní DOM element. Jak již název napovídá, jedná se o kliknutí na požadovaný element. Blok kódu 9 zobrazuje možné užití tohoto příkazu.

```
cy.get(„#button“).click()
cy.contains(„Submit“).click()
```

Programový kód 9: Ukázka použití click (vlastní práce autora)

6.5.3.1.7 type

Další akční příkaz, který se, stejně jako *click()*, řetězí až za příkazy, které odkazují na DOM element. Pro úspěšné vykonání však musí selektor odkazovat na jeden z těchto elementů: *a[href], area[href], input, select, textarea, button, iframe, [tabindex], [contenteditable]*. Příkaz *type()* podporuje psaní nejen klasického textu, ale také umožňuje provádět akce kliknutí na tlačítko klávesnice, jako je například enter nebo směrové šipky, pokud uživatel umístí požadovanou a podporovanou klávesu do složených závorek. V kódu 10 je příklad použití tohoto příkazu.

```
cy.get(„#form-username“).type(„Jméno Uživatele{enter}“)
```

Programový kód 10: Ukázka použití type (vlastní práce autora)

6.5.3.1.8 select

Příkaz *select()* vybere z nabídky elementu select konkrétní možnost. Tato možnost je přijímána jako parametr. Cypress umožňuje možnosti vybírat podle různých parametrů, jako je text, který se vyskytuje v možnosti, hodnota jeho atributu value, nebo index takové možnosti. Výhodou je, že při vybírání více možností, se tento příkaz nemusí volat vícekrát. Namísto toho je možné jako parametr použít pole hodnot. V kódu 11 je vidět způsob použití tohoto příkazu.

```
cy.get(„#gender“).select(„Muž“)
cy.get(„#gender“).select(„1“)
cy.get(„#gender“).select(0)
cy.get(„#typ-vlasu“).select([„Dlouhé“, „Blonďaté“, „12“, 3])
```

Programový kód 11: Ukázka použití select (vlastní práce autora)

6.5.3.2 Konkrétní použití příkazů

Po seznámení se se základními operacemi, klíčovými pro napsání vlastního testu, můžeme vytvořit jeden ukázkový test pro otestování. Tento test navštíví stránku webovou stránku *cypress.io/app*, kde nejprve odmítne ukládání cookies, poté najde na stránce odkaz pro změnu cookies preferencí a nové preference uloží. Obrázek 2 je výsledkem průběhu bloku kódu 12.

```
describe('Vlastní test', () => {
  before(() => {
    cy.visit("https://cypress.io/app")
  })
  it('Změna nastavení cookies', () => {
    cy.get('div[role="dialog"] button.osano-cm-denyAll').click()
    cy.get('div.footer-secondary').contains('Cookie Preferences').click()

    cy.get('[for="osano-cm-drawer-toggle--category_PERSONALIZATION"]')
      .should('be.visible').click()
    cy.get('button.osano-cm-save').click()
  })
})
```

Programový kód 12: Ukázka jednoduchého testu (vlastní práce autora)

Obrázek 2: Výsledek jednoduchého testu (vlastní práce autora)

7 Další automatizované testovací nástroje

Cypress není jediným testovacím nástrojem, ale má i své konkurenty. Podle článku [16] je v současné době takových alternativ nespočetné množství. Pro tuto práci byly proto vybrány jen jedny z aktuálně nejznámějších a nejpoužívanějších. Z obrázku 3 byly vybrány pouze nástroje podporující testování webových aplikací.

Product	Katalon	Selenium	appium	TestComplete	cypress
Application Under Test	Web/API/ Mobile/Desktop	Web	Mobile (Android/iOS)	Web/Mobile/ Desktop	Web
Supported platform(s)	Windows/ macOS/ Linux	Windows/ macOS/ Linux/Solaris	Windows/ macOS	Windows	Windows/ macOS/ Linux
Setup & configuration	Easy	Coding Required	Coding Required	Easy	Coding Required
Low-code & Scripting mode	Both	Scripting Only	Scripting Only	Both	Scripting Only
Supported language(s)	Java & Groovy	Java, C#, Python, JavaScript, Ruby, PHP, Perl	Java, C#, Python, JavaScript, Ruby, PHP, Perl	JavaScript, Python, VBScript, JScript, Delphi, C++, C#	JavaScript
Advanced test reporting	✓	✗	✗	✗	✓
Pricing	Free and Paid	Free	Free	Paid	Free and Paid
Ratings & Reviews (Gartner)	**4.4/5** **740 reviews**	**4.5/5** **443 reviews**	4.4/5 90 reviews	4.4/5 45 reviews	4.6/5 27 reviews

Obrázek 3: Porovnání testovacích nástrojů [16]

7.1 Katalon

Katalon je platforma založená na Seleniu a Appiu a nabízí více služeb

7.1.1 Katalon Studio

Autoři článku [11] uvádí, že velkou výhodou tohoto nástroje je no-code možnost psaní testů, tj. pokud se člověk nevyzná v programování, stačí mu pro napsání testu pouze nahrát postup, jakým chce konkrétní stránku testovat. Program si pak sám

postup převede do kódu a podle něj následně automaticky testy opakuje. Pro zkušenějšího programátora nabízí studio i low-code nebo full-code variantu psaní testů – to znamená, že si test může plně naprogramovat v podporovaném jazyce, kterým je Java a Groovy.

7.1.2 Katalon TestOps

Jak zmiňuje dokument [12], tato služba v kombinaci s Katalon studiem nabízí efektivní spolupráci testovacího týmu pro ještě lepší a rychlejší testování. Mimo jiné také poskytuje detailní reporty jednotlivých testů a jejich podrobnou analýzu.

7.1.3 Katalon TestCloud

Díku tomuto nástroji, podle dokumentu [13], může uživatel spustit test virtuálně na jakémkoliv podporovaném operačním systému a v jakémkoliv podporovaném prohlížeči. To znamená, že uživatel nemusí mít vytvořené vlastní testovací prostředí.

7.1.4 Katalon Runtime Engine

Jak uvádí dokument [14], KRE (= Katalon Runtime Engine) doplňuje předchozí nástroj o možnost rychlejší a flexibilnější správy provádění testů. Díky KRE lze plánovat automatické spouštění testů.

7.2 Selenium

Selenium je, jako jeho předchozí alternativa, platforma, která obsahuje více nástrojů.

7.2.1 Selenium WebDriver

Podle autorů dokumentu [15] tento nástroj nahradil starší Selenium RC (= Remote Control) kvůli jeho nedostatkům:

- RC je pomalejší z důvodu používání javascriptového programu, nazývaného selenium core.
- RC ke svému běhu vyžaduje běh serveru.
- RC nepodporuje Ajaxové aplikace.

WebDriver, také známý jako Selenium 2.0, vznikl hlavně za účelem zlepšení podpory pro novější webové aplikace. Navzdory vylepšením má však také svoje nedostatky, kterými jsou např. nemožnost zachytit obrazovku v době, kdy nastala chyba, nebo postrádá funkci generování výsledků testu.

7.2.2 Selenium IDE

Selenium IDE (Integrated Development Environment) je podle článku [16] rozšíření prohlížeče Chrome nebo Firefox, které umožňuje vytvářet a spouštět testy podobně, jako Katalon Studio. Uživatel programem nahraje požadovaný průběh testu, který se převede do funkčního kódu v případě Selenia se však uživatel neobejde bez znalosti programování. Stejně jako WebDriver trpí svými nedostatky, jako je nemožnost zachytit snímek obrazovky a chybí mu také generace výsledků testů. Stejně jako u WebDriveru lze ale tyto nedostatky odstranit přidáním podporovaných plug-inů.

7.2.3 Selenium Grid

Tento nástroj umožňuje paralelizaci testů s cílem snížit celkový čas potřebný k testování. Například pokud chce uživatel spustit 15 testů, kdy každý test trvá 45 vteřin, a nepoužívá Grid, celkem testování trvá více než 11 minut. Pokud ale uživatel Grid využije, a jednotlivé testy rozdělí mezi 15 virtuálních zařízení, trvá celé testování 45 vteřin, viz blíže v článku [17]. Urychlení testování však není jediná výhoda, dále totiž Selenium Grid nabízí možnost testování na různých operačních systémech a prohlížečích zároveň. Z 15 virtuálních zařízení může být například 5 s nejnovější verzí Firefoxu, 3 se starší verzí Chromu a zbytek s různými verzemi prohlížeče Safari.

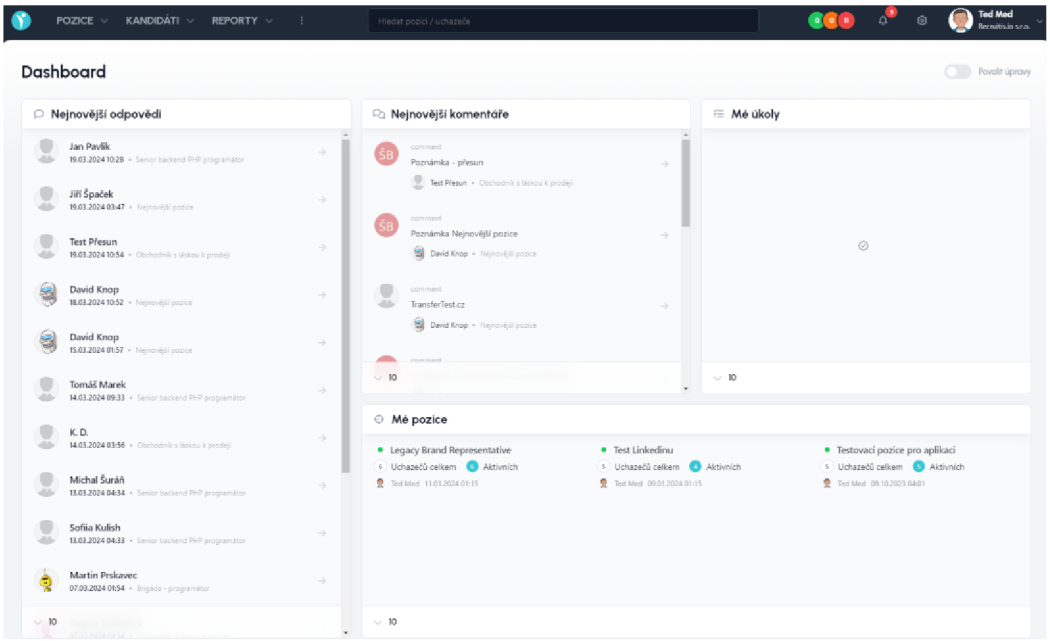
7.3 TestComplete

Tento nástroj je pouze jeden s mnoha produktů společnosti SmartBear, která se zaměřuje mimo jiné na automatizaci testování. Nabízí možnost psaní testů jak pro uživatele bez znalostí programování díky funkci „record and replay“, která nahraje průběh manuálního testování uživatele a poté dokáže proces opakovat, tak pro programátory díky podpoře několika programovacích jazyků. TestComplete

po otestování poskytuje snímky obrazovky z míst, ve kterých test selhal, a dokonce umožňuje nahrát videozáznam celého průběhu testování. Díky propojení ekosystému produktů od SmartBear lze přidat další klíčové funkce, jako je paralelní testování, testování ve virtuálních prostředích, nebo rozpoznání vizuálních defektů, které neodhalí klasické funkční testování, díky AI.

8 Systém Recruitis.io

Recruitis.io je náborový software a mobilní aplikace, vyvíjený společností Recruitis.io s.r.o. Jedná se o jeden z mnoha takzvaných ATS systémů, které se na trhu nabízí. Cílem takových systémů, jak uvádí web [27], je vylepšit, zjednodušit a zefektivnit proces náboru nových uchazečů o pracovní místo, a usnadnit a modernizovat práci HR oddělení.



Obrázek 4: Ukázka úvodní strany systému (vlastní práce autora)

Na webu [28] jsou popsány jednotlivé funkce pro usnadnění práce náborovým zaměstnancům, mezi které patří například import kandidátů ze sítě Linkedin, plánování pohovorů s kandidáty a synchronizace s externími kalendáři, vytváření vlastních odpovědních formulářů, nebo také nová mobilní aplikace.

8.1 Webová aplikace z technického hlediska

Webová aplikace Recruitis.io se skládá z frontendu a backendu. Tyto dvě části spolu komunikují díky REST API. Z důvodu vyvíjení všech částí systému se jedná o full-stack aplikaci. Frontend využívá převážně javascriptový framework VueJS2 (https://v2.vuejs.org/). Protože se ale stále přidávají nové funkce, aktualizují se i nástroje a jazyky, nové části systému již používají VueJS3 (https://vuejs.org/) v kombinaci s Typescriptem. Backend je napsán v programovacím jazyce PHP s využitím frameworku Symfony (https://symfony.com/). Verze PHP se také postupně aktualizuje, jen kvůli udržení kompatibility ne tak často. Pro ukládání všech dat se používá zejména relační databáze MariaDB (https://mariadb.org/). Zpětná vazba pro uživatele tohoto systému je velmi důležitá, ať už v podobě reportů nebo analytiky. Pro tyto účely využívá systém Elasticsearch (https://www.elastic.co/elasticsearch). V systému lze také hledat například kandidáty v databázi uchazečů podle jejich jména, emailu, nebo textu v poznámce k uchazeči. Pro takovéto fulltextové vyhledávání slouží Sphinx. Z důvodu práce s velmi citlivými daty všech uživatelů systému a uchazečů je celá infrastruktura kvůli zabezpečení a ochraně těchto dat hostována v privátním cloudu.

8.2 Myšlenka automatizace testování pro systém Recruitis.io

I přes to, že v Recruitis.io s.r.o. jsou lidské zdroje pro případné otestování provedených změn systému, není ani zdaleka možné neustále testovat každou změnu a její případné negativní dopady na ostatní části aplikace. Z tohoto důvodu bylo ve společnosti Recruitis.io vytvořeno oddělení, které se věnuje automatizaci právě tohoto testování. Aby bylo takové testování maximálně efektivní, vyvstaly v průběhu manuálního testování následující požadavky, které byly splněny právě testovacím nástrojem Cypress, a následně byly implementovány samotným autorem práce:

- Používat jednoduchý nástroj, který umožní snadné psaní a následnou správu testů.
- Možnost integrace s Gitlab CI/CD, kvůli zakomponování testů přímo do procesu vývoje aplikace – pokud by měla vyjít nová verze aplikace,

nejprve na ní musí úspěšně projít testy a až poté se změna projeví. Pokud se tak nestane, vrátí se změna zpět příslušnému vývojáři a proces se opakuje.

- Běh testů na lokálním prostředí pro konzistentnost dat a ochranu hlavní databáze.
- Ze začátku bylo cílem testovat hlavní kategorie systému, kterými jsou pozice a kandidáti, konkrétně přidání a odebrání pozice a kandidáta. V průběhu času se ale požadavky začaly rozšiřovat a postupně se přidávalo testování nastavení, a práce s kandidáty jako je kopírování na jinou pozici a přesouvání do různých stavů flow.
- Detailní reporting selhaných testů byl dalším hlavním požadavkem. V průběhu testování se však postupně začaly objevovat nedostatečně popsané chyby, a snímky obrazovky v momentě selhání neodpovídaly realitě, proto, kvůli vyšším nárokům na zpětnou vazbu ze selhaných testů, byly přidány pluginy zaměřené na reporting.

V průběhu testování však nebyly splněny všechny požadavky, mezi které patří například tyto:

- Otestovat práci s externími kalendáři. Protože Cypress umožňuje testovat pouze jednu doménu, a to jen v jednom okně prohlížeče, není možné otestovat například přidání externího kalendáře Google, který lze přidat pouze skrze vlastní přihlašovací okno, které má jinou doménu.
- Spouštět testy jak celé aplikace od začátku do konce (end-to-end), tak testy jednotlivých částí systému. I přes to, že Cypress nabízí více možností testování, testovací tým usoudil, že by psaní většího množství testů nebylo efektivní, proto se rozhodl psát pouze end-to-end testy, které budou procházet postupně celou aplikaci. Z tohoto důvodu také na sebe nějaké dílčí testy navazují. Důvodem nesplnění tohoto požadavku je také to, že při používání testovací databáze, která se po každém testu resetuje, nebylo možné otestovat určité scénáře.

Celkově byl však nástroj Cypress nejvhodnějším nástrojem a byl zvolen pro testování. Návrhu a implementaci jednotlivých testů se bude věnovat zbytek této bakalářské práce.

9 Implementace automatizovaných testů pro webovou aplikaci Recruitis.io

Tato kapitola pojednává nejprve o vybraném účtu, používaném pro testování, dále o scénáři testování a struktuře vybraných testů, a na závěr se kapitola zaměřuje na samotnou implementaci testů.

9.1 Testovací účet

Webová aplikace Recruitis.io nabízí šest základních typů účtů uživatelů, které se liší právy pro čtení a úpravu různých sekcí a částí softwaru. Tímto rozdělením se dá dobře strukturovat vnitřní uspořádání jednotlivých firem, které si tento systém pořídí. Pro testování byl vybrán účet ***Personalista admin***, který nabízí největší množství práv, aby bylo možné otestovat všechny dostupné nabízené funkce. V případě, že by byl vybrán účet s menšími právy, nebylo by možné testovat například editaci uživatelů systému, nebo práce s vlastními (= custom) položkami, které jsou pro funkci systému velmi důležité. Na obrázku 5 je porovnání dvou typů účtů s nejméně omezenými právy a jednotlivé rozdíly v nich.

	Personalista admin	Hlavní personalista
Má přístup k zadávání inzerátů	✓	✓
V seznamu pozic vidí i pozice ostatních.	✓	✓
V detailu pozice vidí i uchazeče ostatních.	✓	✓
Má přístup k inzerátům druhých	✓	✓
Má přístup ke všem žádánkám	✓	✓
Vidí SLA semafor kolegů	✓	✓
Má přístup ke všem emailovým šablonám a může je upravovat	✓	✓
Má přístup ke všem kandidátům v databázi	✓	✓
Má přístup k editaci firemních nastavení	✓	×
Má přístup k editaci uživatelů	✓	×
Může nastavovat a upravovat custom položky (štítky, formuláře, flow náboru, důvody zamítnutí, atd.)	✓	×
Má přístup pouze k nasdíleným inzerátům a uchazečům	×	×
Může vidět manažerský report.	✓	✓
Vidí kompletní kartu kandidáta	✓	✓
Může s kandidátem provádět akce	✓	✓

Obrázek 5: Rozdíly mezi právy dvou uživatelů (vlastní práce autora)

9.2 Scénář testování

Jak již bylo zmíněno, testuje se celá aplikace od začátku do konce, tedy end-to-end. Existuje proto pouze jeden testovací scénář, podle kterého pak běží naprogramovaný test, který sestává z více logicky oddělených úseků. Tyto úseky pak obsahují jednotlivé testy. Jeden dílčí test většinou testuje právě jeden případ použití aplikace, jako je například přidání kandidáta, nebo úprava jedné sekce nastavení. Struktura testů je znázorněna v tabulce 1.

Kategorie	Název	Předmět testu
Positions	addPosition	Přidání pozice
	delPosition	Odebrání pozice
Candidates	addCandidate	Přidání kandidáta
	delCandidate	Smazání kandidáta
	copyCandidate	Zkopírování kandidáta na jinou pozici
	moveCandidate	Přesunutí kandidáta na jinou pozici
Calendly	addMeet	Přidání šablony meetingu
	shareLink	Sdílení šablony uchazeči
	planMeeting	Naplánování schůzky
	sharedLinkMeeting	Speciální schůzka
Calendar	addEventAsUserA	Přidání eventu do kalendáře
	planCalendlyAsUserB	Naplánování meetingu přes uživatele s jinými právy
	addNewCalendlyAsUserB	Přidání šablony meetingu přes uživatele s jinými právy
Settings	company, notifications, emailTemplates, addresses, users, graphics, positionEditor, answerForms, jobOfferTemplates, flow, nps, rejectReasons, filterableItems, tags, answerSources, channels, gdprTemplates, checklists	Tato sekce projde celé nastavení.

Kategorie	Název	Předmět testu
Filters	gdpr, special, rejected, sources, positions, wage, tags	Tato sekce testuje filtrování v databázi uchazečů

Tabulka 1: Kategorie a jednotlivé testy (vlastní práce autora)

Struktura testu je tak díky tomuto členění přehledná a srozumitelná. Protože, jak již bylo zmíněno dříve, bylo rozhodnuto, že spouštět testy jednotlivých komponent systému nemá ve vývojovém flow význam, všechny testy se spouští postupně za sebou, aby bylo ověřeno, že aplikace funguje správně ve všech svých částech, a že jedna změna neovlivnila aplikaci tam, kde by ovlivněna být neměla. Některé testy na sebe dokonce navazují, zejména to jsou testy přidání kandidátů, které navazují na test přidání pozice, nebo testy sdílení šablony meetingu uchazeči, které musí následovat až za testy přidání šablony meetingu.

Takový způsob testování ale ze zkušeností není úplně vhodný, hlavně z důvodu nekonzistentnosti návazných testů v případě selhání testu, na kterém další závisely. Takový test, který byl spuštěn pomocí příkazu ‚*npx cypress run*' a jeden z jeho dílčích testů selhal, musel běžet až do úplného konce, aby byly uloženy snímky obrazovky v momentě chyby. V průběhu testování byl ale nalezen plug-in, který umožnil probíhající test v prostředí GitLab CI/CD zastavit v momentě, kdy jeden z dílčích testů selhal, což vyřešilo problém s navazujícími testy a velice zrychlil proces debugování v případě selhání – nebylo již nutné čekat do úplného konce testu pro zjištění důvodu chyby.

9.2.1 Struktura kompletního testu

Předchozí tabulka nezobrazuje strukturu kompletního testu. Kompletní test se skládá i z pomocných částí, kam patří například přihlášení uživatele. Celkově má pak test následující strukturu:

I. Vymazání lokální paměti a cookies prohlížeče

II. Session[1] (= relace) pro přihlášení uživatele do systému.
    a. Vytvoření relace.
    b. Obnovení relace.
III. Provedení dílčího testu.
IV. Odhlášení uživatele ze systému.
V. Vymazání lokální paměti a cookies prohlížeče.

Krok I. se provádí pouze jednou, a to na úplném začátku. Kroky IV. a V. se naopak provádí na konci a každý z nich také jen jednou. Vytvoření relace z kroku II.a. se spustí opět pouze jednou, a to před prvním dílčím testem neboli krokem III. Před každým dalším testem se poté provádí krok II.b., ve kterém se již jednou vytvořená a uložená relace obnoví.

Obnova relace však neprobíhá před úplně všemi dílčími testy, například po testu *addNewCalendlyAsUserB*, kdy se místo uložené relace s přihlášeným uživatelem a přihlásí uživatel B, se mažou všechny uložené relace, a to z důvodu častých chyb při obnově přihlášení uživatele a u následujících testů.

Kroky I. a V. není nutné v testu explicitně přidávat, protože, jak je zmíněno v článku [9] v sekci Test Isolation, Cypress má defaultně nastavené, že před každým testem vymaže kontext prohlížeče. Z tohoto důvodu se také používá příkaz *cy.session()* pro uložení a udržení stavu přihlášeného uživatele.

9.2.2 Struktura jednotlivých testů

V předchozí části lze dobře vidět struktura všech testů dohromady. V této části bude detailněji rozebrána struktura jednotlivých testů, konkrétně testu vytvoření inzerátu (pozice) *addPosition* a přidání uchazeče (kandidáta) do databáze *addCandidate* jedním ze čtyř testovaných způsobů.

[1] Podle [28] dokáže relace ukládat do mezipaměti a obnovovat soubory cookies, úložiště localStorage a data z relace za účelem obnovení konzistentního kontextu prohlížeče mezi jednotlivými testy.

Pořadí	Událost	Požadovaný cíl
0	Vymazání localStorage a cookies.	Vymazány cookies a localStorage.
1a.1	Navštívení přihlašovací stránky.	Načteny všechny potřebné komponenty.
1a.2	Vyplnění přihlašovacího formuláře a následné přihlášení tlačítkem.	Přihlášení a načtení domovské stránky systému, uložení tohoto stavu do relace.
2	Navštívení stránky pro vytvoření pozice */editor/create*.	Otevřena požadovaná stránka a načteny všechny komponenty.
3.1	Vyplnění dostupných položek formuláře pro vytvoření pozice.	Všechny required a dostupné položky jsou vybrány a doplněny.
3.2	Přejití na následující stránku pro další nastavení vytvářené pozice.	Načtena další stránka s formulářovými prvky.
3.3	Vyplnění dalších položek formuláře pro vytvoření pozice.	Všechny required a dostupné položky jsou vybrány a doplněny.
3.4	Přejití na finální stránku přidání pozice.	Načtena poslední požadovaná stránka s výběrem míst pro sdílení inzerátu.
3.5	Vybrání míst, kde bude inzerát dostupný online a potvrzení vytvoření.	Úspěšné uložení pozice, status kód odpovědi serveru je mezi 199 a 400.

Tabulka 2: Postup testu addPosition (vlastní práce autora)

Pro porovnání zobrazuje tabulka 3 test přidání uchazeče do databáze uchazečů.

Pořadí	Událost	Požadovaný cíl
1b	Obnovení relace přihlášení.	Obnovení do stavu přihlášeného uživatele, načtení domovské stránky.
2	Navštívení stránky pro přidání uchazeče */talenti*.	Otevřena požadovaná stránka a načteny všechny komponenty.
3.1	Kliknutí na tlačítko pro přidání uchazeče do databáze.	Načten popup pro přidání kandidáta.
3.2	Vyplnění všech požadovaných polí formuláře.	Kandidát má vyplněné všechny požadované prvky.

Pořadí	Událost	Požadovaný cíl
3.3	Uložení kandidáta do databáze pomocí tlačítka	Všechny required a dostupné položky jsou vybrány a doplněny.
3.4	Přejití na finální stránku přidání pozice.	Načtena poslední požadovaná stránka s výběrem míst pro sdílení inzerátu.
3.5	Vybrání míst, kde bude inzerát dostupný online a potvrzení vytvoření.	Úspěšné uložení pozice, status kód odpovědi serveru je mezi 199 a 400.

Tabulka 3: Postup testu addCandidate (vlastní práce autora)

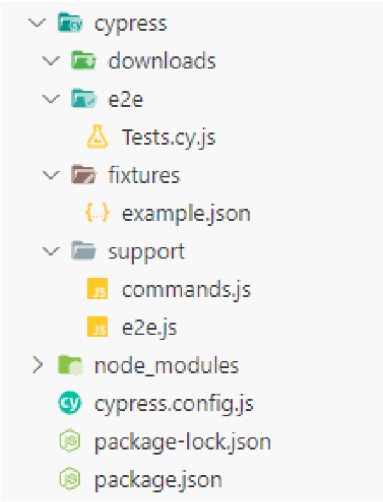
Obecně lze s dosavadními poznatky říci, že mají testy velmi podobnou strukturu. Zásadnější odlišnosti jsou viditelné v kroku 3.1 a krocích následujících, přičemž krok číslo 2 se liší pouze v adrese cílové stránky. Krok 1a a 1b se liší pouze v typu práce s relací. Nultý krok se v celém testování vyskytuje pouze jednou, a to před úplně prvním testem

9.3 Implementace

V této části je již popsána samotná příprava projektu k psaní testů a následné psaní testů.

9.3.1 Lokalizace projektu

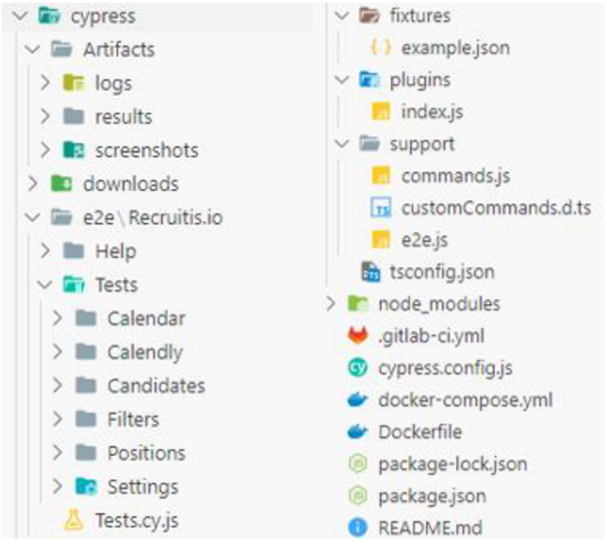
Pro správné nastavení projektu je vhodné nejprve vybrat umístění, ve kterém se bude celý projekt a jeho součásti nacházet. V tomto případě je použit adresář recruitis-cypress. Jeho obsah po spuštění instalačního příkazu, který byl zmíněn v sekci 6.5.1 této práce, po otevření programu Cypress a následném vytvoření testovacího souboru pomocí výběru v aplikaci, je pro představu zaznamenán na obrázku 6.



Obrázek 6: Obsah projektu po prvotním nastavení (vlastní práce autora)

Cypress je hlavní složkou projektu a nachází se v ní všechny potřebné podsložky. Do složky *downloads* se ukládají soubory stažené v průběhu testu, *e2e* je hlavní složka samotných testových souborů a složka *fixtures* obsahuje soubory, které reprezentují fixní testovací data. Poslední zobrazenou složkou je složka *support*, která obsahuje pomocné soubory, jako jsou například vlastní, nebo jen upravené, již existující, příkazy, nebo úpravy globální konfigurace cypressu.

Toto rozdělení slouží pouze pro ukázku struktury po prvotním nastavení. Rozdělení v obrázku 7 je již reálná podoba adresáře pro testování.



Obrázek 7: Aktuální obsah adresáře (vlastní práce autora)

Pro přehledné oddělení souborů generovaných po neúspěšném testu byla vytvořena složka *Artifacts*, která obsahuje v dceřiných složkách logy všech vykonaných příkazů posledního selhaného testu, soubor typu .xml pro generování přehledného reportu testu v prostředí Gitlab CI/CD a snímky obrazovky. Jak je patrné, videa zde svoji složku nemají, a to hlavně z důvodu zrychlení celého testu. Složka *downloads* je jen pozůstatkem prvotní instalace a není tedy při testování využívána. Ve složce *e2e* jsou všechny testovací soubory, jako jsou dílčí testy, které jsou pouze ve formátu .js a jeden spouštěcí soubor s příponou .cy.js, který shromažďuje všechny dílčí testy. Složka *fixtures* je, stejně jako *downloads*, nepoužívaným pozůstatkem po prvotní instalaci. *Plugins* obsahují soubor, který spouští přídavné pluginy a *support* obsahuje soubor s upravenou konfigurací a vlastní vytvořené příkazy. Následně adresář obsahuje konfigurační soubor, ve kterém jsou specifikovaná různá nastavení programu. Další soubor, kterým je *.gitlab-ci.yml*, je zaměřený na prostředí Gitlab CI/CD a specifikuje spouštění samotného testu v tomto prostředí. Docker soubory jsou určené pro práci s docker imagem.

9.3.2 Psaní testů

Podle struktury ze sekce 9.2.1 je nejprve připraveno hrubé rozdělení souboru, který bude později obsahovat reference na dílčí testy a jako jediný se bude spouštět. Vytvoří se tedy jeden spouštěcí soubor, v tomto případě se jedná o *Tests.cy.js*.

9.3.2.1 Společný blok

Dále je potřeba si rozmyslet, v jakých bodech se jednotlivé testy liší a jaké body mají naopak společné. Po zjištění, že se jednotlivé části liší od bodu, ve kterém test navštěvuje testovanou stránku, se vytvoří bloky, které jsou společné pro všechny testy, jak je znázorněno na bloku kódu 13.

```
before(() => {
    Cypress.session.clearAllSavedSessions()
    cy.clearAllCookies()
    cy.clearAllLocalStorage()
    cy.clearAllSessionStorage()
});
beforeEach(() => {
    cy.session('Log in', () => {
        logIn(jmeno, heslo)
    })
});
after(() => {
    logOut()
    cy.clearAllCookies()
    cy.clearAllLocalStorage()
    cy.clearAllSessionStorage()
});
```

Programový kód 13: Společné kroky pro jednotlivé testy (vlastní práce autora)

Před celým testováním se spouští první část tohoto kódu, konkrétně blok *before*, a naopak na konci se provádí blok *after*. I přes to, že Cypress před spuštěním testování localStorage, cookies a paměť relace maže automaticky, při opakovaném spouštění testu v prostředí Cypress občas docházelo k chybám právě s uloženými cookies, proto jsou před i po testování všechna uložená data explicitně mazána. Jak již bylo také zmíněno, Cypress maže kontext prohlížeče i mezi jednotlivými testy, a proto je před každým dílčím testem přihlašování do systému vyřešeno pomocí relace v bloku *beforeEach*. Funkce login přijímá jako parametry jméno a heslo uživatele. V tomto případě jsou parametry uložené do proměnných *jmeno* a *heslo*. Dále již následují samotné testy.

9.3.2.2 Jednotlivé testy

Po definování společné části přichází na řadu jednotlivé testy, které jsou, jak již bylo zmíněno dříve, rozdělené do logických celků, kterými jsou sekce *Candidates*, *Calendly*, *Calendar*, *Settings* a *Filters*. Jednotlivé sekce tedy ještě obsahují dílčí testy, a některé sekce jsou ještě více rozděleny do menších celků, jako je například rozdělení sekce *Candidates* do části *Adding 4 candidates, deleting one* a části *Copying and moving candidates*.

Jednotlivé dílčí testy, jsou oddělené nejen logicky do sekcí v hlavním testovacím souboru, ale také v samotných souborech. Jsou tedy vytvořeny soubory, jako jsou například *addCandidate* pro přidání kandidáta čtyřmi různými způsoby, přičemž každý jeden způsob má svůj vlastní test, nebo planMeeting, který také obsahuje více testů, které jsou všechny zaměřené na naplánování schůzky. Tyto jednotlivé testy v různých souborech se používají jako export funkce, které se následně importují a používají v hlavním testovacím souboru. Na úryvku kódu 14 je znázorněno rozdělení testovacích sekcí.

```
// // Candidates:
context('Add position', () => {
  addPos(position)
})
context('Candidates', () => {
  Candidates(position)
})
context('Delete position', () => {
  delPos(position)
})
// // Calendly:
context('Calendly', () => {
  Calendly(name2, "Kateřina Dvořáková", "Pavel Měřínský")
})
// // Calendar:
context('Calendar', () => {
  addEventAsUserA()
  planCalendlyAsUserB()
  addNewCalendlyAsUserB()
})
// // Settings:
context('Settings', () => {
  Settings()
})
// // Filters:
context('Filters', () => {
  Filters()
})
```

Programový kód 14: Jednotlivé testy rozdělené do sekcí (vlastní práce autora)

Jak lze v kódu vidět, *position* a *name2* jsou proměnné. Druhá z nich je exportována ze souboru *Candidates* a importována do hlavního testovacího souboru. Do proměnné *position* je uloženo jméno pozice, které je vždy náhodně generováno pomocí javascriptové knihovny faker-js[2]. Nyní je již možné začít jednotlivé sekce plnit samotnými testy.

Celé testování začíná vytvořením testovací pozice, která se v dalších krocích testování používá pro vytváření a přesouvání kandidátů a následně se tato pozice smaže. Blok kódu 15 zobrazuje zkrácenou verzi testu přidání pozice.

[2] Faker-js je knihovna, která generuje falešná (ale reálně vypadající) data.

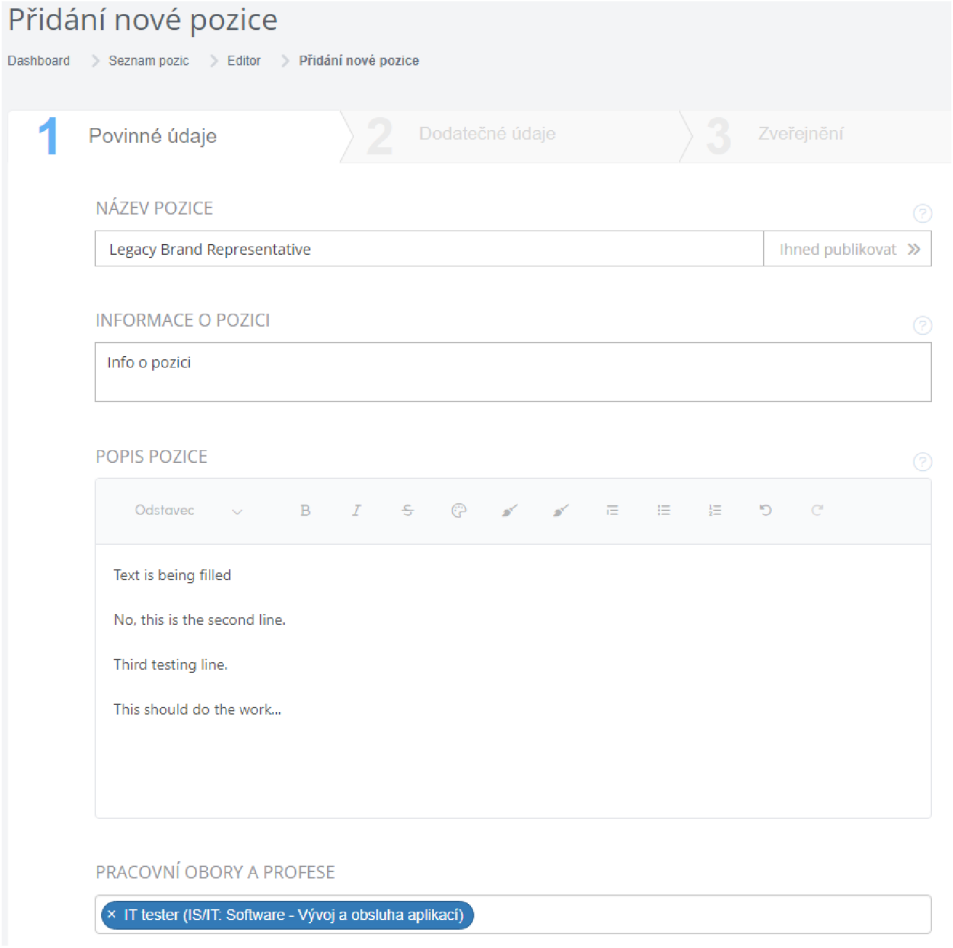
```
export function addPos(position) {
  it('Creates a position', () => {
    cy.task('log', 'Creates a position')
    inter('post', '/zadavatel/editor/finish', 'posCreated')
    inter('post', '/zadavatel/novinky/check', 'pageLoad')
    inter('post', '/zadavatel/editor/save-part', 'savePart')
    inter('post', '/zadavatel/inzerat/get_social_thumbs/*', 'photoThumb')
    cy.visit('/editor/create')
    // Page #1
    cy.get('#title').should('exist').and('be.visible').type(position)
    ifExists('#editor-wrapper', '#frm_details.comment',
    () => {
      cy.get('#frm_details.comment').type('Info o pozici')
    }, () => {})
    cy.get('.ck-editor__main > .ck').type('Text is being filled\nNo, this
is the second line.\nThird testing line.\nThis should do the work...')
    cy.get('#s2id_autogen2').click().type('IT Tester{enter}')
    cy.get('#frm_text_language').select(0)
    cy.get('#step-mandatorypage-editor > .btn-red').click()
    cy.get('#editor-wizard ').should('have.class', 'active')
    // Page #2
    cy.get('#salary_range_2).should('be.visible').type(25000)
    cy.get('select[name="salary_currency"]').select(1)
    cy.get('#salary-unit').select('month')
    cy.get('[name="details.education"]').select(3)
    cy.get('#content #step-optionalpage-editor > div > .btn').click()
    cy.get('#editor-wizard').should('have.class', 'active')
    // Page #3
    cy.get('.toggle > .rounded-4x').should('be.visible').click()
    cy.get('.btn-finish').click()
    waitForAlias('posCreated')
  });
}
```

Programový kód 15: Test přidání pozice (vlastní práce autora)

V první části testu, ještě před navštívením testované stránky, si lze všimnout vlastní funkce *inter()*, jejímž úkolem je přidat serverovému volání typu POST, které má určitou url adresu, jasný alias, aby bylo možné takové volání v testu odchytit a vyčkat na vrácení odpovědi ze serveru.

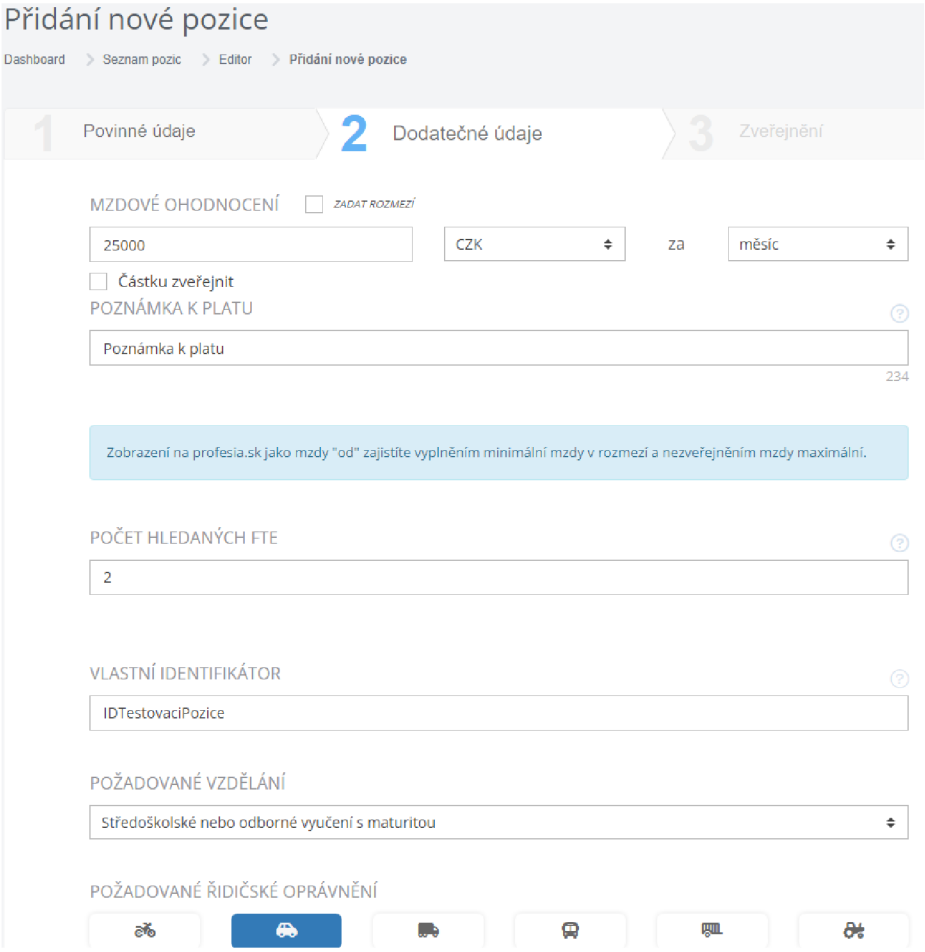
Za přiřazením aliasů jednotlivým voláním následuje otevření a načtení stránky pro přidání inzerátu. Další vlastní funkcí je *ifExists()*, která v případě, že hledaný

element existuje, provede požadovanou akci, a v opačném případě akci jinou, nebo žádnou. Dalším krokem je vyplnění povinných údajů, na obrázku 8 je zobrazena část takových údajů.



Obrázek 8: Vyplněné povinné údaje pro vytvoření pozice (vlastní práce autora)

Jak si lze na snímku všimnout, název pozice vypadá reálně, ale je náhodně vygenerovaný knihovnou faker-js. Po vyplnění všech povinných údajů lze přejít na druhou stránku s dodatečnými údaji. V tomto případě textové pole pro vyplnění informací o pozici existuje, proto je vyplněno. Na obrázku 9 je znázorněna vyplněná i druhá stránka.



Obrázek 9: Vyplněné dodatečné údaje pro vytvoření pozice (vlastní práce autora)

Po vyplnění i druhé stránky lze přejít na tu poslední, která slouží pro vybrání míst, na kterých bude inzerát zveřejněn. Lze vybrat, že bude inzerát pouze interní a zveřejněn vůbec nebude, nebo je možné publikovat na různých pracovních portálech, nebo vlastních publikačních kanálech. Po vybrání míst ke zveřejnění lze již inzerát uložit.

Stejným způsobem se postupně implementují další testy v pořadí, v jakém jsou seřazeny jednotlivé sekce.

9.3.3 Spouštění testů

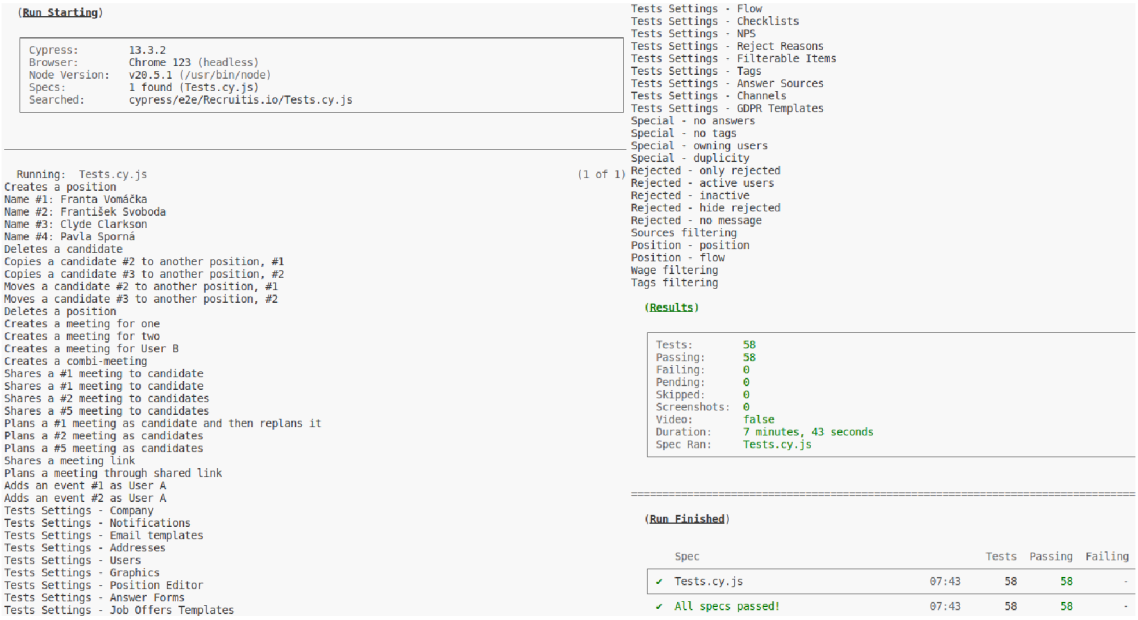
Testy jsou spouštěny výhradně v prostředí GitLab CI/CD, a to pomocí příkazu v kódu 16. Tento příkaz se nachází v souboru .gitlab.ci spolu s dalšími nastaveními prostředí a způsobu chování testu a jeho součástí v něm.

```
TZ=Europe/Prague npx cypress run --spec
./cypress/e2e/Recruitis.io/Tests.cy.js --headless --browser chrome
--reporter junit
```

Programový kód 16: Spouštění testu v prostředí GitLab CI/CD (vlastní práce autora)

Cypress dokáže rozpoznat spustitelné soubory díky jejich příponě .cy.js, a proto pokud není explicitně určena cesta k požadovanému souboru, spustí se postupně všechny soubory s touto příponou. Testování probíhá běžně na třech prohlížečích, Chrome, Edge a Firefox. Proto se do spouštěcího příkazu přidává možnost „browser" s konkrétním prohlížečem. V současné době však testování na prohlížeči Firefox z důvodu neznámé chyby neprobíhá.

V případě jakékoliv opravy nebo přidání nových testů se však spouští testování i pomocí příkazu, který je zobrazený v programovém kódu 4 v sekci 6.5.2. Díky tomu tester vidí přesný postup testu, a nejen jeho výsledek, což mu výrazně usnadňuje a urychluje ladění chyb. Na obrázku 10 je vidět úspěšný výsledek testování.



Obrázek 10: Úspěšný výsledek testování (vlastní práce autora)

9.3.4 Zajímavá řešení

Prvním zajímavým řešením je pomocná funkce *ifExists()*. Cypress vždy postupuje v průběhu testu příkaz po příkazu, ale v případě funkce *if()*, nebo také *for()* a *while()*,

vyhodnocuje tyto bloky v momentě spuštění dílčího testu, nelze tedy tyto funkce použít. Proto byla vytvořena pomocná funkce, která blok *if()* obaluje do bloku, který se spouští až v požadované chvíli. Tato funkce je zobrazena v programovém kódu 17.

```
export function ifExists(parentElement, searchableElement,
onExistsCallback, onNonExistsCallback) {
  cy.get(parentElement).then(() => {
    if (Cypress.$(searchableElement).length) {
      onExistsCallback();
    } else{
      onNonExistsCallback();
    }
  })
}
```

Programový kód 17: Funkce ifExists (vlastní práce autora)

Dalším zajímavým řešením je práce s Iframy. Protože Iframe je webová stránka, vložená do jiné, Cypress s ní neumí pracovat. K tomu, aby bylo možné testovat například boční panely v systému Recruitis, slouží právě funkce *getIframeBody()*. V konkrétním testu se pak tato funkce řetězí přímo za „cy“ a za ní běžně následuje funkce *within()*, která zaručí, že následující příkazy operují v úrovni Iframu. Samotná funkce je vidět na programovém kódu 18.

```
Cypress.Commands.add('getIframeBody', () => {
  cy.log('getIframeBody')

  return cy
  .get('iframe.flex-item', { log: false })
  .its('0.contentDocument.body', { log: false }).should('not.be.empty')
  .then((body) => cy.wrap(body, { log: false }))
})
```

Programový kód 18: Funkce getIframeBody (vlastní práce autora)

Dalším, ne už tolik zajímavým a vhodným řešením, ale spíše řešením, které stojí za zmínku, je vybírání časů při vytváření šablony schůzky. Protože se aplikace načítá asynchronně, občas se stalo, že test selhal. Selhání bylo zapříčiněno právě chybou při načtení seznamu dostupných časů. Test nejprve vybere datum pro šablonu schůzky, a následně vybírá čas, od a do kterého je v dané datum možné schůzku

naplánovat. Cypress se snaží z elementu select vybrat čas, ale ve chvíli, kdy s tímto elementem chce reagovat, dojde nečekaně k asynchronnímu načtení a test selže. Z tohoto důvodu bylo nutné přidat čekací dobu 2 s, jak je vidět na úryvku kódu 19.

```
cy.wait(2000)
cy.get("label.select > select").eq(0).should('be.visible').as('timeFrom1')
cy.get('@timeFrom1').select('12:00').then(() => {
  cy.get("label.select > select").eq(0).should('have.value', "12:00")
})
cy.wait(2000)
cy.get("label.select > select").eq(1).should('be.visible').as('timeTo1')
cy.get('@timeTo1').select('16:00').then(() => {
  cy.get("label.select > select").eq(1).should('have.value', "16:00")
})
```

Programový kód 19: Vyřešení asynchronního načítání (vlastní práce autora)

9.4 Problémy při psaní

Největším problémem, který v tomto testování ještě není vyřešen úplně ideálně je asynchronní načítání. Velkým problémem jsou právě drobná obnovení stránky, protože ne vždy je v testu něco, čeho se chytit, jako například zobrazeného načítacího kolečka, nebo odpovědi serveru. Používaným řešením je zatím konstantní čekací doba v rozpětí od dvou do pěti sekund, podle místa použití. Toto řešení však není úplně vhodné, a to z důvodu prodloužení testovací doby.

S předchozím problémem se úzce pojí další nedostatek, kterým je hledání elementů na stránce. Konkrétně se jedná o hledání elementů, které jsou vytvořené frameworkem VueJS (https://vuejs.org/), protože právě ty jsou od sebe odlišitelné většinou jen přesnou pozicí na stránce a nedají se nalézt přes jejich ID. Z tohoto důvodu jsou však na přání testerů do těchto komponent přidávány atributy *data-cy*, díky kterým je pak nalezení velmi snadné, a hlavně již nezávisí na přesné struktuře stránky. Na bloku kódu 20 je znázorněn rozdíl mezi starším hledáním bez atributu a novějším hledáním s atributem *data-cy*.

```
cy.get('#meeting-overview > div:nth-child(2) > div > div')      //Stary
cy.get('[data-cy="form-title"]')                                //Novy
```

Programový kód 20: Rozdíly s atributem data-cy (vlastní práce autora)

10 Shrnutí výsledků

Automatizace testování webové aplikace Recruitis.io bylo klíčovým krokem nejen pro zefektivnění testovacího procesu a tím i celého vývoje softwaru, ale také vedlo k odstranění základních chyb, které se do té doby mohly vyskytnout v produkčním prostředí. I přes drobné nedostatky, které byly uvedeny, probíhá testování v programu Cypress, který byl vybrán jako nejvhodnější nástroj pro tento způsob použití.

Díky možnosti propojení s prostředím GitLab probíhá testování výhradně v tomto prostředí, kde se testy spouští v režimu bez uživatelského rozhraní. Pro úpravu stávajících testů, nebo pro vytvoření testů zcela nových, se využívá i lokální prostředí s testy spouštěnými v režimu s uživatelským rozhraním.

Jednotlivých testů bylo napsáno celkem 60, ale z důvodu nefunkčnosti dvou z nich se jich aktuálně používá pouze 58, jak lze vidět na obrázku 10. Podle stejného obrázku lze také poznat, jaké části systému se testují. Testy ověřují správnou funkčnost přidání a odebrání pozice, práci s kandidáty, práci s pohovory, různé změny nastavení a filtraci v databázi uchazečů. Aktuálně jsou všechny testy úspěšné, jak si lze všimnout na obrázku 10.

Celkově testy trvají v prostředí GitLab průměrně 10,5 minuty v případě, že všechny testy prošly úspěšně. V opačném případě, díky plug-inu pro předčasné zastavení testu, mohou testy probíhat i výrazně kratší čas. Rychlost testování je závislé na typu prostředí a množství dostupných prostředků pro testování, protože v lokálním prostředí trvají testy průměrně 8 minut.

Průběžně je vynakládána snaha očistit aktuální testy o nedostatky a problémy dosud zjištěné. Nejvíce se nyní dbá na jednoduchost a přesnost selektorů, které se užívají pro hledání elementů na stránce. Z tohoto důvodu se přidávají atributy *data-cy* do elementů, které zatím nelze najít jinak než přes přesnou strukturu stránky.

11 Závěry a doporučení

Tato bakalářská práce měla za cíl seznámit čtenáře s typy a metodami testování, a konkrétněji popsat implementaci testování webové aplikace Recruitis.io. Výstupem praktické části této bakalářské práce je soubor testů, aktuálně používaných pro automatizované testování webové aplikace Recruitis.io.

V první části byly teoreticky rozebrány způsoby a možnosti testování s procesem automatizace. Dále byl popsán nástroj určený k automatizaci, konkrétně se jednalo o nástroj Cypress. Následně byly popsány alternativy tohoto nástroje a jejich výhody, případně nevýhody.

Testy, které byly zdrojem čerpání dat v praktické části, jsou již nyní nedílnou součástí vývojového procesu společnosti Recruitis.io s.r.o. a stejně jako celý vývoj probíhají v prostředí GitLab pro co nejlepší zasazení do procesu.

12 Seznam použité literatury

[1] Druhy testování v procesu vývoje SW. Testování software [online]. [cit. 20.04.2023]. Dostupné z: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=18:druhy-testovani-v-procesu-vyvoje-sw&catid=3:zaklady&Itemid=11

[2] KHORIKOV, V. Unit Testing Principles, Practices, and Patterns. Spojené státy Americké, Manning. 2020. ISBN 9781638350293.

[3] HLAVA, Tomáš. Druhy, Typy a Kategorie Testů | Testování softwaru. Testování softwaru. [online]. 21.08.2011 [cit. 20.04.2023]. Dostupné z: http://testovanisoftwaru.cz/category/druhy-typy-a-kategorie-testu

[4] GUSS, Robert, LUCAS, Ely, JAFFRE, Paul. Cypress Documentation. Cypress Documentation [online]. Copyright © 2023 Cypress.io. All rights reserved. [cit. 20.4.2023]. Dostupné z: https://docs.cypress.io

[5] STEPHENS, John. Jaký je rozdíl mezi funkčním a nefunkčním testováním. [online]. 2021 [cit. 21.04.2023]. Dostupné z: https://cs.strephonsays.com/what-is-the-difference-between-functional-and-nonfunctional-testing

[6] JAIN, Mahak. Differences between Black Box Testing vs White Box Testing – GeeksforGeeks. GeeksforGeeks | a computer science portal for geeks [online]. [cit. 21.04.2023]. Dostupné z: https://www.geeksforgeeks.org/differences-between-black-box-testing-vs-white-box-testing

[7] GILLIS S., Alexander. What Is Static Testing?. Purchase Intent Data for Enterprise Tech Sales and Marketing – TechTarget [online]. [cit. 21.04.2023]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/static-testing

[8] HAMILTON, Thomas. What is Dynamic Testing? Types, Techniques & Example [online]. [cit. 21.04.2023]. Dostupné z: https://www.guru99.com/dynamic-testing.html

[9] GUSS, Robert; LUCAS, Ely; JAFFRE, Paul. Core Concepts [online]. Copyright © 2023 Cypress.io. All rights reserved. [cit. 26.4.2023]. Dostupné z: https://docs.cypress.io/guides/core-concepts

[10] O'REGAN, Gerard. Software Testing. V: Introduction to Software Quality [online]. Cham: Springer International Publishing, 2014, strany. 119–134 [cit.

17.10.2023]. ISBN 9783319061054. Dostupné z: doi: https://doi.org/10.1007/978-3-319-06106-1_7
[11] Katalon Studio. Katalon [online]. 2015 [cit. 2024-01-26]. Dostupné z: https://katalon.com/katalon-studio
[12] Katalon TestOps. Katalon [online]. 2015 [cit. 2024-01-26]. Dostupné z: https://katalon.com/testops
[13] Katalon TestCloud. Katalon [online]. 2015 [cit. 2024-01-26]. Dostupné z: https://katalon.com/testcloud
[14] Katalon Runtime Engine. Katalon [online]. 2015 [cit. 2024-01-26]. Dostupné z: https://katalon.com/katalon-runtime-engine
[15] GOJARE, Satish; JOSHI, Rahul a GAIGAWARE, Dhanashree. Analysis and Design of Selenium WebDriver Automation Testing Framework. [online]. Procedia Computer Science. 2015, roč. 50, s. 341-346. ISSN 18770509. [cit. 2024-01-26]. Dostupné z doi: https://doi.org/10.1016/j.procs.2015.04.038
[16] Top 15 Automation Testing Tools. Katalon [online]. 2015 [cit. 2024-01-26]. Dostupné z: https://katalon.com/resources-center/blog/automation-testing-tools
[17] When To Use Grid. Selenium [online]. 2015 [cit. 2024-01-26]. Dostupné z: https://www.selenium.dev/documentation/grid/applicability
[18] Patton, R. Software Testing. Indianapolis: Sams Publishing, 2005, ISBN 0-672-32798-8
[19] Using Cypress for email or SMS testing. Mailosaur [online]. 2023 [cit. 2024-01-26]. Dostupné z: https://mailosaur.com/docs/frameworks-and-tools/cypress
[20] Learning how Visual AI works. Applitools [online]. 2023 [cit. 2024-01-26]. Dostupné z: https://applitools.com/tutorials/guides/getting-started/learning-how-it-works
[21] MCMILLAN, Taryn. Manual Testing vs Automated Testing: Key Differences. TestRail [online]. 2023 [cit. 2024-01-28]. Dostupné z: https://www.testrail.com/blog/manual-vs-automated-testing
[22] VUOLLET, Phil. What Is Test Automation? a Simple, Clear Introduction. Testim [online]. 2023 [cit. 2024-01-28]. Dostupné z: https://www.testim.io/blog/what-is-test-automation

[23] EHMER, Mohd a KHAN, Farmeena. a Comparative Study of White Box, Black Box and Grey Box Testing Techniques. Online. International Journal of Advanced Computer Science and Applications. 2012, roč. 3, č. 6. ISSN 2158107X. [cit. 2024-03-06]. Dostupné z: https://doi.org/10.14569/IJACSA.2012.030603

[24] Cypress Installation [online]. 2024 [cit. 2024-03-06]. Dostupné z: https://docs.cypress.io/guides/getting-started/installing-cypress

[25] Opening the App [online]. 2024 [cit. 2024-03-18]. Dostupné z: https://docs.cypress.io/guides/getting-started/opening-the-app

[26] Command Line [online]. 2024 [cit. 2024-03-18]. Dostupné z: https://docs.cypress.io/guides/guides/command-line

[27] Recruitis.io - Co je ATS? [online]. 2018 [cit. 2024-03-21]. Dostupné z: https://recruitis.io/blog/clanek/co-je-ats/

[28] Session [online]. 2024 [cit. 2024-03-28]. Dostupné z: https://docs.cypress.io/api/commands/session

[29] Faker. Online. 2024 [cit. 2024-04-04]. Dostupné z: https://fakerjs.dev/guide

13 Seznam obrázků

14 Seznam tabulek

15 Seznam (programových) kódů

16 Přílohy

1. Soubor s vytvořenými testy je nahraný v systému eVŠKP.

Upozornění: Vytvořené testy jsou z právních důvodů upravené tak, aby je nebylo možné spustit. Součástí testů nejsou lokální prostředí ani přihlašovací údaje, které jsou pro běh testů zásadní.

Zadání práce z IS/STAG

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2022/2023

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení:	**Tadeáš Jirsa**
Osobní číslo:	**I2100216**
Adresa:	V Uličce 44, Pardubice – Popkovice, 53006 Pardubice 6, Česká republika
Téma práce:	Automatizované testování webové aplikace Recruitis v nástroji Cypress
Téma práce anglicky:	Automated testing of Recruitis web application in Cypress
Jazyk práce:	Čeština
Vedoucí práce:	Ing. Martina Husáková, Ph.D. Katedra informačních technologií

Zásady pro vypracování:

Cíl práce:
Cílem bakalářské práce je analyzovat možnosti a způsoby testování webových aplikací s důrazem na využití nástroje Cypress k tomuto účelu. Nástroj Cypress bude využit pro vytvoření automatizovaných testů pro webovou aplikaci Recruitis.io.
Osnova:
Úvod
Kategorie testování
Metody testování
Testovací nástroj Cypress
Další testovací nástroje
Implementace automatizovaných testů pro webovou aplikaci Recruitis.io
Shrnutí
Závěr a doporučení
Literatura

Seznam doporučené literatury:

KHORIKOV, V. Unit Testing Principles, Practices, and Patterns. Spojené státy Americké, Manning. 2020. ISBN 9781638350293.
GUSS, Robert, LUCAS, Ely, JAFFRE, Paul. Cypress Documentation. *Cypress Documentation* [online]. Copyright © 2023 Cypress.io. All rights reserved. [cit. 20.4.2023]. Dostupné z: https://docs.cypress.io/
O'REGAN, Gerard. Software Testing. V: Introduction to Software Quality [online]. Cham: Springer International Publishing, 2014, strany. 119–134 [cit. 17.10.2023]. ISBN 9783319061054. Dostupné z: doi:10.1007/978-3-319-06106-1_7

Podpis studenta: Datum:

Podpis vedoucího práce: Datum:

© IS/STAG, Portál – Podklad kvalifikační práce , jirsata1, 17. dubna 2024 18:13