**Univerzita Hradec Králové**
**Fakulta informatiky a managementu**
**Katedra managementu**

# Tvorba testovací aplikace, automatizace a optimalizace testovacího procesu

Diplomová práce

Autor: Bc. Jakub Karl
Studijní obor: IM2

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové duben 2022

Prohlášení:

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

V Hradci Králové dne 29.4.2022 Jakub Karl

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Ing. Filip Malý, Ph.D. za metodické vedení práce, vstřícný přístup a cenné rady.

# Anotace

Tématem této diplomové práce je tvorba testovací aplikace, automatizace a optimalizace testovacího procesu. V teoretické části jsou představeny především technologie využívané pro vývoj aplikace. Těmito technologiemi jsou především na klientu běžící programovací jazyk JavaScript, na serveru běžící Node.js, NoSQL databáze MongoDb, též modelovací technologie UML a BPMN a mnoho dalších technologií. Důležitou součástí je též představení celého testovacího procesu a jeho částí. V praktické části jsou pak tyto dílčí části sestaveny v logický celek, který znázorňuje návaznost v testovacím procesu.

Hlavním cílem diplomová práce je vývoj software (konkrétně webové aplikace) usnadňující testovací proces nazývaný „produktové testování“, vývoj automatických testů a celková optimalizací celé workflow. Aplikace je navržena tak, aby pokrývala výše zmíněné části celé flow a bylo ji možné implementovat na server firmy Avast, kde bude dále využívána při testování.

# Annotation

**Title: Creating a test application, automation, and optimization of the test process**

The topic of this diploma thesis is “Creating a test application, automation and optimization of the test process”. Technologies such as programming language JavaScript, server running Node.js, NoSQL database MongoDb and also modelling technologies UML and BPMN and many others are all described in the theoretic part of this work. Very important part of the theoretic part is the description of the whole testing process and its parts. In the practical part, these parts are then connected into logically related unit, which shows the continuities in the testing process.

The main goal of my diploma thesis is the development of the software (more precisely, a web application) that should facilitate testing process called “product testing”, development of automated tests and overall optimization of that

workflow. The application is designed to cover all the above-mentioned parts of the entire flow. Application should be implemented into company Avast's server and used by for testing purposes.

# Obsah

# 1 Cíl práce

Hlavním cílem diplomové práce s tématem „Tvorba testovací aplikace, automatizace a optimalizace testovacího procesu“ byla analýza testovacího procesu, který by se dal nazvat „testování produktového listování“ (tj. testování nových produktů v interních, externích systémech, e-shopu neboli košíku apod.) a následné vytvoření aplikace, která by měla celý proces zjednodušit, zkrátit a celkově zoptimalizovat. S tím je spojena i tvorba automatických testů.

Aplikace se zaměřuje na všechny dílčí části testovacího procesu. Jedná se o fullstack webovou aplikaci. Pojem fullstack značí, že aplikace je tvořena front-endovou částí, v tomto případě psanou především v programovacím jazyce Javascript, a back-endovou částí, pro níž je využita technologie známá jako Node.js. Aplikace využívá též databázový systém, konkrétně jednu z nejpopulárnějších NoSQL databází – MongoDB. Nad rámec těchto tří částí obsahuje práce ještě automatické testy psané v E2E testovacím frameworku Cypress založeném též na Javascriptu. Testy neslouží pro otestování webové aplikace, ale pro testování především e-shopu firmy Avast.

Celá práce by měla přinést komplexní náhled na celou problematiku již zmíněného testovacího procesu, ať už z pohledu manuálních testů, či testů automatických a dalších testovacích nástrojů a postupů.

## 1.1 Metodika zpracování

Metodika zpracování vychází z analýzy odborné literatury, odborných článků a dalších zdrojů zabývajících se problematikou testování, vývojem software a optimalizačními technikami procesů. Nejprve bylo potřeba seznámit se s problematikou daného tématu, vymezit pojmy s tím spojené a následně nabité teoreticé znalosti aplikovat do praktické části práce. Cílem teoretické části práce není podrobná analýza všech možných řešení pro optimalizaci procesu a vývoj testovacího software, neboť těch nespočetně. Teoretická část vymezuje pouze technologie využité v této práci.

# 2 Teoretická část

## 2.1 O firmě Avast

Avast software s.r.o. je česká firma známá především vývojem antivirového programu. Zabývá se však kybernetickou bezpečností celkově. Kromě antivirového software (verze zdarma i placené) vyvíjí a prodává firma i produkty pro čištění počítače (CleanUp), software proti trackování (AntiTrack), VPN (SecureLine), program k práci s PC bateriemi (Battery Saver) a mnoho dalších. Firma se nezaměřuje však pouze na ochranu počítačů běžných uživatelů (tj. consumer), ale též vyvíjí produkty pro business účely (SMB, tj. small and medium-sized business). [1]

Vznik této nadnárodní společnosti se datuje k roku 2010, ačkoliv její předchůdce firma Alwil vznikla již v roce 1988. V tento rok vznikl původní program, ze kterého se později vyvinula další navazující řešení. Zakladateli firmy jsou Pavel Baudiš a Eduard Kučera. V roce 2016 došlo k odkupu další velké firmy se zaměřením na počítačovou bezpečnost, AVG Technologies. Od té doby spravuje firma nejen značku Avast, ale i AVG. Kromě těchto dvou zmíněných brandů, jsou s firmou Avast spojeny značky CCleaner a HMA (Hide My Ass). [2]

**Obr. 1 Logo firmy Avast**
Zdroj: https://press.avast.com/cs-cz/media-materials

## 2.2 Optimalizace procesu

Optimalizace procesu je implementace strukturovaných metod, strategií, disciplín a taktik ke zlepšení konkrétního procesu v rámci parametrů projektu. Existuje mnoho způsobů, jak lze upravit proces tak, aby objektivně fungoval lépe, než fungoval. Vyjmutí kroku, odstranění kroku nebo přepsání kroku v procesu jsou zjednodušené příklady, které mohou pomoci zefektivnit pracovní postup. Obecně je možné optimalizaci procesu definovat též jako proces, který je složený z následujících kroků: [3]

- Identifikace: jedná se o určení procesu, který musí projít optimalizací. Pokud je proces příliš nákladný nebo způsobuje nespokojenost zákazníků, pak je pro úspěšné fungování jakékoli optimalizační metody klíčové určení příčiny těchto obav. [3]
- Přehodnocení: přehodnocení identifikovaného procesu a kritické zkoumání jeho účelu v rámci celkového pracovního postupu. Zjištění, jak dlouho proces trvá, než se dokončí, nebo zda existuje lepší způsob, jak jej provést, jsou jen dvě z několika možností, jak přehodnotit identifikovaný proces. [3]
- Implementace: proces, který byl identifikován, že je třeba jej optimalizovat a následně byl i předělán je třeba posléze implementovat, tj. v tomto případě nahradit proces starý procesem novým. Po dokončení implementace nového nebo upraveného procesu je dobré zanalyzovat výsledky a podle potřeby provést úpravy, aby se zajistilo, že vše bude fungovat tak, jak bylo zamýšleno. [3]
- Automatizace: nový proces je posléze třeba zautomatizovat v rámci celého pracovního postupu, aby bylo možné vyhodnotit, zda z něj vyplývají požadované závěry, jako je snížení nákladů, zvýšení výroby nebo prevence chyb. Před automatizací je důležité se ujistit, že proces skutečně funguje. [3]
- Monitorování: proces je po jeho úspěšné integraci třeba monitorovat. Díky monitoringu je možné identifikovat zjistit, zda funguje v časově delším horizontu správně a případně identifikovat nové oblasti pro zlepšení. Pomáhá též určit případné kritické problémy, které je třeba vyřešit, aby pracovní postup pokračoval správně i nadále. [3]

Z optimalizace procesu by mělo vyplývat mnoho výhod. Pokud po optimalizaci není možní požadované výhody vysledovat, nebyla optimalizace provedena správně. Zlepšeními mohou být například: [4]

- snížení rizik neefektivního procesu nahrazením za nový
- větší konzistence – odstranění redundancí
- zlepšení kvality výstupu z procesu
- zefektivnění operací
- zlepšení správy zdrojů
- úspora peněz
- zvýšení produktivity
- minimalizace budoucích chyb
- omezení problémů

### 2.2.1 Optimalizační techniky

#### 2.2.1.1 Metoda PDSA

Jedná se o čtyřdílný cyklický systém, který pomáhá zlepšovat kvalitu procesu a tím docílit optimalizace podnikání. PDSA je slovní zkratka 4 anglických slov – plan, do, study, act. Každé ze zmíněných slov představuje důležitou fázi v rámci celé PDSA metody. [5]

- Plan – zmapování a definování úspěchů, kterých mají být dosaženy
- Do – otestování potenciálních změn v malém a omezeném měřítku
- Study – prostudování výsledků metody a zjištění jejich užitečnosti
- Act – implementace změn ve větším měřítku

#### 2.2.1.2 Metoda Process Mining

Metoda Process Mining je složeninou technik, které zahrnují prvky data science (datové vědy). Metoda je založena na přebírání dat z event logu (protokol

událostí) a následné analýze akcí členů týmu v rámci podnikání, tj. přezkoumání kroků, které tito členové podnikali, aby splnili povinnosti či dosáhli kýženého cíle. Shromážděná data je pak možné převést na použitelné poznatky a pomoci tak identifikovat problémy a optimalizovat potřebné procesy. [6]

#### 2.2.1.3 DMAIC metoda

DMAIC je stejně jako v případě PDSA metody zkratka anglických slov, tentokrát však pěti. Jedná se o slova define, measure, analyze, improve a control. DMAIC metoda je daty řízená strategie složená z následujících kroků: [5]

- Define – definování procesů vyžadujících optimalizaci
- Measure – měření a identifikace toho, jak proces funguje
- Analyze – analýza toho, jak lze proces optimalizovat
- Improve – samotné zlepšení procesu
- Control – kontrola budoucího výkonu

#### 2.2.1.4 BPO project management metoda

Optimalizační metoda projektového řízení BPO je definována následujícími kroky: [5]

- Initiating - schválení fáze, procesu nebo projektu jako součásti celkové iniciativy
- Planning – nejprve je třeba definovat či předefinovat cíle a poté vybrat nejlepší akce, které jich pomohou dosáhnout
- Executing - koordinace a spolupráce s lidmi, kteří pomohou provést plán nebo strategii
- Controlling – zajištění, že každý stanovený cíl je splněn. Je toho docíleno pravidelným sledováním a měřením pokroku a přijímáním nápravných opatření podle potřeby
- Closing - formální uzavření projektu, jakmile metoda dosáhne svého účelu

#### 2.2.1.5 Value stream mapping metoda

Tato metoda se opírá mapu toku hodnot. Jedná se o grafické znázornění různých materiálů, dat a informací, které proudí v rámci projektu nebo iniciativy. Primárním cílem je přinášet služby a produkty zákazníkům nebo koncovým uživatelům. Tato metoda je užitečná, protože díky ní lze dosáhnout následujících výsledků: [5]

- identifikace míst, kde se plýtvá zdroji a následné odstranění těchto míst
- získání cenného náhledu na rozhodovací procesy a procesní toky
- stanovení měřitelných a realistických cílů pro zlepšování procesů
- určení oblastí, které potřebují zlepšení

#### 2.2.1.6 Metoda Kaizen

Optimalizační metoda Kaizen může pracovat ve spojení s metodou PDCA. Jedná se o techniku, která pravidelně zlepšuje všechny obchodní funkce. Výhodami této metody jsou: [5]

- podpora rychlejších dodávek a bezpečnostních opatření
- zlepšení pracovní spokojenosti a produktivity týmu
- zlepšení každého obchodního procesu
- zlepšení kvality produktů a hodnocení zákaznické podpory

#### 2.2.1.7 DMADV metoda

Optimalizační metoda DMADV se zaměřuje na to, aby pomohla zvýšit úroveň kvality i po počáteční optimalizační iniciativě. Tato metoda přináší dramatickou změnu tím, že zcela nahrazuje starý proces procesem novým a vylepšeným. Seznam výhod této metody může zahrnovat: [6]

- zvýšení příjmů a zisků
- zlepšení hodnocení a spokojenosti zákazníků a klientů

- Snížení počtu chyb, které mohou nastat
- Vývoj zcela nových produktů a procesů

#### 2.2.1.8 SIPOC analyzační metoda

Metoda SIPOC je užitečná pro organizování shromážděných dat o zákaznících a produktech zapojených do procesů. Tato metoda může být užitečná z několika důvodů, včetně: [6]

- identifikace všech relevantních aspektů, které je třeba zlepšit
- definice složitých projektů, kterým je následně dát účet jejich použití
- pochopení toho, jak by měl proces fungovat

### 2.2.2 SWOT analýza

SWOT analýza je akronym složený z počátečních písmen slov strengths, weaknesses, opportunities, threads. SWOT je technika strategického plánování a strategického řízení, která pomáhá osobě nebo organizaci identifikovat silné (S) a slabé (W) stránky, příležitosti (O) a hrozby (T) související s obchodní konkurencí nebo plánováním projektů. Někdy se nazývá situační hodnocení nebo situační analýza. Tato technika je určena pro použití v přípravných fázích rozhodovacích procesů a lze ji využít jako nástroj pro hodnocení strategické pozice organizací. Je určena k identifikaci vnitřních a vnějších faktorů, které jsou příznivé a nepříznivé pro dosažení cílů podniku nebo projektu. Uživatelé SWOT analýzy často kladou otázky a odpovídají na ně, aby vytvořili smysluplné informace pro každou kategorii a identifikovali tím svou konkurenční výhodu. SWOT byla popsána jako osvědčený nástroj strategické analýzy. [7]

## 2.3 Testování

Testování softwaru je nepostradatelnou součástí vývoje softwaru. Cílem testování není dokázat správné fungování programu, ale nalézt, v ideálním případě, všechny chyby testovaného programu. Pojem testování software lze definovat výčtem činností, které zahrnuje. Kromě hlavní činnosti, tedy verifikace programu, obsahuje testování další činnosti jako jsou statické analýzy, revize dokumentace, inspekce kódu či též testování požadavků. Úkolem testerů je v podstatě destrukce daného softwaru či procesu. Každý tester musí předvídat neznalost uživatele a z toho důvodu by se měl snažit program jakkoliv rozbít, aby se chyby daly včas opravit a uživatel již nemohl program tak snadno rozbít. Z testování vyplývá nespočetné množství výhod. Těmi jsou například: [8]

- snížení nákladů na vývoj – chyba objevena v dřívější fázi projektu je mnohem méně nákladná na opravu
- test měří a podporuje kvalitu a výkon produktu
- zlepšení procesu vývoje
- snadnější přidávání nových funkcí
- lepší UX (user experience) a tím se i zvyšuje i lepší veřejné mínění o brandu
- a mnoho dalších

### 2.3.1 Testovací proces

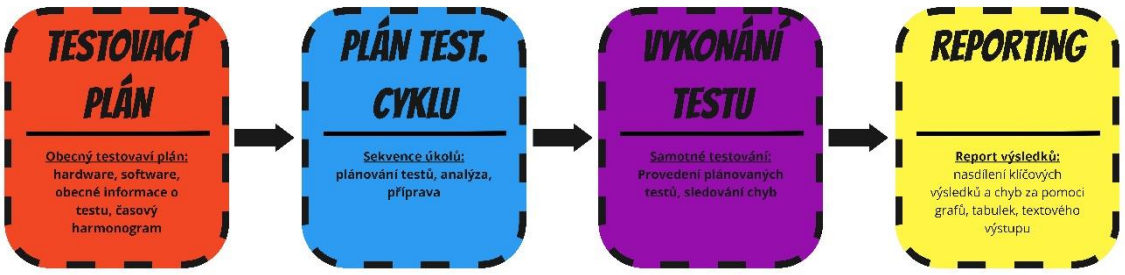
Testovací proces je možné zjednodušeně rozdělit na čtyři části (viz. obr 2). Celý proces začíná vytvořením obecného testovacího plánu. V něm by mělo být obsaženo stanovení cílů, kterých chceme dosáhnout. Dále popisuje, jaký hardware a software se bude testovat a na jakém testovacím místě se bude testovat, a udává časový harmonogram testů. [9]

Testovací cyklus je sekvence úkonů, které musí proběhnout od začátku do konce testování. Těmito činnostmi je například plánování testů, stanovení částí, které je třeba testovat, příprava a analýza dat i samotného testu či příprava

nástrojů pro lepší průběh testu. Kromě výše zmíněných aktivit je potřeba zkontrolovat požadavky na produkt. [9]

Následuje samotné vykonání testů. Testy mohou mít mnoho podob. Základním rozdělením je na testy manuální a automatické. Jak již vypovídají názvy obou kategorií, v případě manuálního testování provádí tester krok za krokem podle specifikace testovacích scénářů. Automatické testy jsou pak prováděny testovacím softwarem. [9]

Celý testovací proces je zakončený reportingem výsledků. Úkolem testera je chyby pouze najít, v lepším případě i se zjištěním důvodu, proč chyba nastala. Není však na testerovi oprava chyb. Z tohoto důvodu je třeba vývojáři detailně popsat problém, aby jej mohl co nejefektivněji opravit. Tomu se říká reporting výsledků. [9]




**Obr. 2 Testovací workflow**
Zdroj: Vlastní zpracování

Testování většinou probíhá ve více iteracích. Iteracím (číslováno od 1) s indexem 2 a výše se říká „retestování“. Aby mohl být celý proces ukončen, je třeba, aby bylo dodrženo několik následujících kritérií: [10]

- všechny testovací případy byly provedeny
- všechny testovací případy dopadly úspěšně
- neexistují nevyřešení incidenty jakéhokoliv charakteru
- počet nalezených vad je již rovno, v ideálním případě, nule
- předdefinované pokrytí bylo dosaženo

### 2.3.2 Dokumentace v testování

Dokumentace je neoddělitelnou součástí v podstatě jakéhokoliv procesu. V případě testování tomu není jinak. Hlavními důvody tvorby dokumentace je usnadnění testování, sdílení vědomostí mezi zaměstnanci a uchování těchto znalostí v případě odchodu zaměstnance, který jako jediný měl potřebné znalosti. [11]

Lidský mozek není neomylný a nelze v něm udržet všechny informace. I z tohoto důvodu je dobré dokumentaci tvořit. Některé druhy testů mohou být každodenní pracovní náplní testera, u jiných však může být časová diference mezi nutností provést daný typ testu značně delší. V takovém případě není nic snazšího než takovou dokumentaci dohledat a řídit se jí. Je velmi pravděpodobné, že databáze s dokumenty bude značně obsáhlá, a proto by měl každý dílčí dokument obsahovat jednoznačný identifikátor, podle nějž je snadné dokument dohledat. [11]

Vytvořením dokumentace však celý proces nekončí. Každý dokument je třeba udržovat stále aktuální. Pokud dojde ke změně v testovací flow, je třeba tuto změnu uvést i do příslušné dokumentace. V případě neudržování aktuální dokumentace může docházet k chybám v testech a ty jsou pak nefunkční a je třeba složitě dohledávat správné řešení. V horším případě budou testy vyhodnoceny jako úspěšné, nebudou nalezeny žádné chyby, přestože tam jsou. [11]

Není přesně stanovena obecná forma, které je třeba se držet při psaní dokumentů, je však možné v rámci firmy či týmu nějakou formu dodržovat. Můžeme však uvést základní dokumenty, které je dobré mít pro kvalitní proces testování: [11]

- Analýza rizik procesu testování softwaru
- Plán testování softwaru
- Testovací případ (Test case)
- Testovací scénář (Test scenario)
- Závěrečné hodnocení průběhu testování softwaru

### 2.3.3 Test management

Test management je pojem pro veškeré činnosti zaštiťující řízení procesu testování. I když může být test management někdy považován za blokující, existuje řada výhod, které nelze ignorovat. To platí zejména pro týmy, které chtějí zlepšit svou efektivitu a vyhnout se nákladným neúspěchům. Zlepšením efektivity a snížením plýtvání v procesu testování, pomáhá test management lépe stanovit priority a zároveň zkrátit čas, který týmy věnují problémům po dodání softwaru. Test management má několik důležitých úkolů, jako následující: [12]

- prioritizace testů
- kontrola dodržování předpisů
- rozhodnutí, kdy produkt doručit
- snížení duplicit v datech
- lepší pokrytí manuálními testy
- zvyšování spolupráce

### 2.3.4 Programování řízené testy

Programování řízení testy (angl. Test Driven Development) je technika založené na opačném postupu, než je při programování a testování zvykem. Prvním krokem není programování, ale sepsání testů, čímž je vlastně vytvořena i definice požadavků na danou funkcionalitu, jež má být naprogramována. Tyto testy jsou pak spuštěny. Očekávaným výsledkem je neúspěch testů, z důvodu že kód ještě není vytvořen. Jedná se i jakousi formu kontroly, že by funkcionalita byla již vyvinuta dříve. Až po napsání a spuštění testů dochází k samotnému programování. Cílem této fáze je pouze úspěšný průchod všemi testy při jejich dalším spuštění. Nekladou se zde velké nároky na efektivitu a „eleganci“ napsaného kódu. Díky úspěšným testům je zajištěno splnění definovaných požadavků. Poslední fází je refaktoring. V této části je kód upravován do co nejpřijatelnější podoby. Tyto fáze jsou opakovány stále dokola. [13]

### 2.3.5 Černá a bílá skříňka

Pojmy Black box a White box, neboli černá a bílá skříňka, známé nejen z oblasti testování, jsou v podstatě založeny na míře informovanosti i testované aplikaci. V případě že tester disponuje omezeným množstvím informací, převážně se jedná o informace z uživatelského pohledu jako je znalost vstupů a výstupů, mluvíme o černé skříňce. Mluvíme-li o bílé skříňce, tak je množství informací poskytnuté testerovi značně větší. V takovém případě má tester přístup i k zdrojovému kódu, což je většinou hlavní náplň testu – otestovat zdrojový kód. Kromě toho by měl mít tester přístup ke všem informacím o aplikaci. [14]

Nelze přesně říci, který přístup je ten lepší, obě varianty mají svoje klady i zápory. Výhodou bílé skříňky je větší komplexita. Testerovi umožňuje zaměřit se na se vnitřní logiku, která není na první pohled patrná a pravděpodobně by nedošlo k její otestování v případě černé skříňky. Na druhou stranu černá skříňka je častějším a jednodušším přístupem. Tester nepotřebuje znalost programovacích jazyků, ani algoritmizace jako v případě White boxu. Ačkoliv, jak bylo řečeno, černá skříňka se vyskytuje častěji než skříňka bílá, tak nejčastějším případem v praxi je skříňka šedá (Grey box). Šedou skříňku můžeme definovat tak, že tester má kromě znalosti vstupů a výstupů i základní znalost vnitřních procesů aplikace. Tester má znalosti o aplikaci nad rámec černé skříňky, ale nedosahuje znalostí černé skříňky. [14]

Proč se takovýmto procesům říká zrovna černá a bílá skříňka? Je tomu tak z toho důvodu, že si aplikaci můžeme představit jako skříňku, jejíž vnitřní obsah je zdrojový kód. Pokud jej neznáme, není pro nás obsah zvenčí viditelný – jedná se o černou skříňku. Bílá skříňka je opakem. [14]

### 2.3.6 Automatizace testování

Automatizované testování přináší mnoho výhod do vývoje software. Takovými výhodami můžou být uspořené prostředky, zvýšení kvality produktu, rychlost testování a především větší přesnost testování a mnoho dalších. Automatické testy se dají napojovat na release jednotlivých verzí produktu, a tím

zajistit její okamžité otestování. Hlavním impulsem pro přechod od manuálního testování k automatizovanému by mělo být časté a opakované testování určitého postupu. Ve chvíli, kdy je třeba určitý manuální test provádět častěji, například jednou týdně (frekvence může být i daleko menší), bylo by dobré se zamyslet, zda ho nelze převést do kódu a spouštět bez zásahu lidského faktoru. Oblast automatizace testů je čím dál populárnější a často vyhledávanou možností jak software testovat. [16]

Hlavním cílem automatizace je časová úspora, usnadnění testování a zefektivnění celého procesu. Pokud je automat správně naprogramovaný, je daleko přesnější než lidské oko, které může spoustu věcí přehlédnout. Dá se obecně říci, že automatizací se snižuje chybovost při testování, a tím se zvyšuje bezporuchovost software. [15]

Automatické a manuální testy se dělí do stejných kategorií, které jsou uvedeny níže v této práci. Některé z nich se pro automatizaci hodí více, některé méně. Do kategorie, které se pro automatizování hodí, spadají především regresní testy, smoke testy a také nefunkční testy. Obecně tedy testy, kterých spouštíme velké množství, jsou neměnné a pokrývají části aplikace, které se často nemění. [16]

### 2.3.7 Rozdělení testů dle ISTQB

ISTQB je zkratka pro „International Software Testing Qualifications Board“. Jedná se o mezinárodní testovací organizaci. Organizace je známá především pro poskytování standardizovaných kvalifikací pro software testery. Dle této testovací certifikační rady se testy dělí do 4 skupin: [17]

- funkční testování
- nefunkční testování
- strukturální testování
- retestování a regresní testování

#### 2.3.7.1 Funkční a nefunkční testování

Jedním možným dělením automatických testů je rozdělení na testy funkční a nefunkční. V případě funkčních testů, jak již název vypovídá, jsou testovány obecně všechny funkcionality aplikace. Je zjišťován správný chod celé aplikace, jsou testovány implementované funkce, požadavky zákazníka, a proto se jedná o velmi důležitou součást testovací části celého systémového vývoje. Tyto testy jsou nejčastěji využívány v integrační, systémové a akceptační úrovni testování a mají zásadní vliv na správné fungování software. [18]

Naopak testy nefunkční jsou často v praxi opomínány, případně nejsou aplikovány v dostatečném rozsahu. Daly by se definovat jako „testy, které nesouvisí přímo s funkcemi aplikace, ale jsou důležité pro její správný chod“. Mohou sem být zařazeny „performance testy“, tedy testy zjišťující výkonnost aplikace v případě jejího přetížení při zpracování většího množství dat či připojení více uživatelů apod. [18]

#### 2.3.7.2 Strukturální testování

Tento způsob testování se od ostatních liší v přístupu k testovanému systému nebo jeho dílčím komponentám. Strukturální testy neřeší, co daný systém vykonává, ale jakým způsobem ji vykonává. Z tohoto vychází i samotný název „strukturální testování“. Je totiž testována struktura systému, ne jeho výstup. Pro strukturální testování je nutné mít větší znalosti v programování a tvorbě aplikací, jelikož při psaní strukturálních testů je zapotřebí číst zdrojový kód. [17]

#### 2.3.7.3 Retestování a regresní testování

Neodlučitelnou součástí testovacího procesu je retestování. Pokud je při testování nalezena chyba a následně opravena programátorem, je třeba ji poté znovu otestovat, tzv. retestovat. Retest se dá nazvat též „konfirmace“. [17]

Pakliže testujeme program, který byl již dříve otestován, avšak následně do něj byly přidány další změny, které se testovaly separátně, mluvíme o regresním testování. Jedná se o zjištění, zda daná úprava testovaná separátně a fungující

správně nezpůsobila problém v jiné části systému. Jedná se tedy též o jistou formou retestování, avšak zde zjišťujeme, zda oprava chyby nezpůsobila chybu jinou. [18]

### 2.3.8 Rozdělení testů podle Luarie Williams

Předcházející členění dle asociace ISTQB je sice mezinárodně uznávané testery z celého světa, avšak toto členění je značně obecné. Následující členění podle Laurie Williams rozšiřuje předchozí dělení a uvádí následující typy testů: [17]

- Unit testy
- Integrační testy
- Systémové testování
- Akceptační testy
- Regresní testy
- Beta testy

#### 2.3.8.1 Unit testy

Unit testy, neboli jednotkové testy, jsou, jak již název vypovídá, testy, které testují pouze jednu jednotku zdrojového kódu. Příkladem může být otestování jedné třídy či metody v případě objektově orientovaného programování. Unit test by se měl zaměřovat pouze jednu danou věc separovanou od zbytku. [17]

Unit testy by měly být vytvořeny velmi brzy po napsání kódu, který má být kontrolován, respektive by se s nimi mělo počítat již ve fázi návrhu aplikace. Je těžké aplikovat jednotkové testy na aplikace, které již nějaký čas běží. V takovém případě je často nutný refaktoring kódu (tedy zlepšení již existujícího kódu) [18]

#### 2.3.8.2 Integrační testy

U tohoto typu testu dochází ke změně oproti předchozím zmíněným testům. Tyto testy již nejsou většinou vytvářeny programátory, ale přichází na řadu testovací tým. Tyto testy bývají občas označovány jako „testy vnitřní integrace“. Tento název je vypovídající. Jedná se primárně o testování komunikace mezi jednotlivými komponentami aplikace, které byly již testovány samostatně např. pomocí unit testů. Pod tímto typem testu se ukrývá i testování komunikace komponent aplikace s operačním systémem, hardwarovým vybavením, na němž je aplikace závislá či na dalším rozhraní jiných systémů. [17]

#### 2.3.8.3 Systémové testy

Dalším druhem testů v celém testovacím procesu jsou systémové testy. Jedná se o nejdůležitější fázi testování, jelikož je systémové testování obsaženo prakticky v každém testovacím procesu. Bez něj by celé testovací flow nedávalo smysl. U těchto testů je již nahlíženo na aplikaci jako na celek. Při tvorbě testovacích scénářů se využívá především uživatelského přístupu. Jsou tedy vytvářeny scénáře, které by měly vystihnout různé chování uživatele. Méně zkušený uživatel může v aplikaci nadělat více nepořádku než užitku, je tedy důležité s tímto počítat a s pomocí systémových testů udělat aplikace tzv. „bullet-proof“. Systémové testy jsou v praxi spouštěny většinou vícekrát. Ve chvíli, kdy test odhalí chybu a tato chyba je opravena, je následně aplikace znovu otestována stejnými testy. Jedná se tedy většinou o poslední testování před předáním výsledného software zákazníkovi. Systémové testy společně s testy integračními se řadí do kategorie „System Integration Tests“ (SIT). [18]

#### 2.3.8.4 Akceptační testy

Na akceptační testy je nahlíženo dvěma způsoby – podle ISTQB a Luarie Williams. ISTQB tvrdí, že tyto testy jsou prováděné přímo zákazníkem, což však Williams vyvrací s vyjádřením, že zákazník nemusí mít dostatečné znalosti a zkušenosti pro správné provedení této testovací fáze. Obecně se však dá říct, že

akceptační testy kontrolují, zda byla splněna všechna kritéria nezbytná pro převzetí produktu zákazníkem. Testy jsou prováděny po úspěšném dokončení všech funkčních, nefunkčních i systémových testů. [17]

#### 2.3.8.5 Regresní testy

Tuto část není třeba více rozvádět z důvodu stejného náhledu Luarie Williams a asociace ISTQB na danou problematiku. Více je popsáno v sekci 2.3.7.3.

#### 2.3.8.6 Beta testy

Beta testování je založeno na sběru dat od uživatelů software. Předpokladem pro uskutečnění beta testování je zveřejnění téměř hotové aplikace uživatelům. Uživatelé pak systém využívají jako jej budou využívat po celkovém dokončení. Je však třeba brát na vědomí, že produkt není zcela dokončený. Uživatelé využívající produkt dávají zpětnou vazbu výrobci software, ten tyto připomínky zpracovává a vyvozuje z nich závěry. [18]

Beta testování má výhodu v tom, že uživatel nemusí mít zdaní, jakými funkcionalitami aplikace disponuje a kde jsou její hranice, tudíž může přijít na chyby a mezery, které by se během jiných testů nepodařilo zjistit. Nevýhodou pak je časová náročnost zpracování často špatně či nedostatečně definovaných chyb. [18]

### 2.3.9 Další typy testů

#### 2.3.9.1 Assembly tests

Tento typ testů je úplně první v pořadí před všemi dalšími testy. Jedná se o testy prováděné přímo programátorem ihned po vytvoření software. V praxi nefunguje toto testování tak, že by si vývojář kontroloval svůj napsaný kód. Uplatňuje se většinou metoda zvaná „test čtyř očí“. V podstatě se jedná o „code review“, tedy kód jednoho vývojáře testuje vývojář jiný. Assembly testy fungují na úrovni kódu - vývojář kontroluje přímo zdrojový kód. Tyto testy jsou často

podceňovány a vývojáři se jimi nezabývají, přestože jsou nejméně náročné a mohou ulehčit spoustu práce do budoucnosti, především testerům. Pokud je v kódu chyba, například i takového ražení, že aplikaci nelze spustit, pak je lepší odhalit tento problém ihned v zárodku před tím, než tester začne studovat danou problematiku a začne vytvářet testovací scénáře či dokonce samotné testování. [17]

#### 2.3.9.2 Smoke testy

Smoke testy jsou prováděny až ve chvíli, kdy je dokončen software, respektive ve chvíli, kdy je plně spustitelný, tedy na konci úrovně integračního testování. Smoke testy jsou krádké a rychlé testy sloužící pro zjištění, zda je aplikace připravena pro další fáze testovaní. Testy se zaměřují pouze na hlavní funkce programu, tedy aby aplikace šla spustit, všechny její části byly správně naimplementovány a vše fungovalo tak, jak má. Smoke testy mívají většinou spíše automatizovanou podobu, nicméně není to podmínkou. [18]

#### 2.3.9.3 Sanity testy

Sanity testy by se do češtiny daly přeložit jako „testy zdravého rozumu“. Je to základní test, který rychle vyhodnotí, zda nějaké tvrzení nebo výsledek výpočtu jsou správná. Jde o jednoduchou kontrolu, zda je vytvořený produkt „racionální“. Smyslem sanity testů je vyloučení určitých tříd zjevně chybných výsledků, nikoli zachytit každou možnou chybu. Výhodou provedení sanity testu je rychlé vyhodnocení základní funkcionality. [19]

Příkladem může být aritmetika. Například při dělení číslem 3 je použití pravidla o dělitelnosti tímto číslem (ověření, že součet číslic v dělenci je též dělitelný číslem 3) v podstatě sanity teste. Test nezachytí každou chybu dělení, ale je to rychlá a jednoduchá metoda k odhalení mnoha možných chyb. [19]

V informatice je sanity test velmi krátkým prozkoumáním funkčnosti počítačového programu, systému, výpočtu nebo jiné analýzy, aby se zajistilo, že část systému nebo metodologie funguje zhruba podle očekávání. [19]

### 2.3.10 Ekonomická stránka testování softwaru

Při vývoji softwaru je vždy třeba vyčlenit určité náklady spojené s testováním programů. Je nutné sepsat plán testování a testovací scénáře, nastavit celé testovací prostředí, ať už z pohledu software, tak hardware, systematicky provádět samotné testy, sledovat zjištěné problémy a musíme odstranit většinu chyb. Často se stává, že jsou nalezeny chyby s nízkou prioritou a je rozhodnuto, že je příliš nákladné chybu opravit, protože je potřeba chybu přepracovat, překódovat nebo jinak odstranit. [36]

Poruchy, které nebyly objeveny a odstraněny před releasem softwaru, jsou v pozdějších fázích často velmi nákladné. Některé z těchto nákladů jsou peněžní a jiné se mohou projevit méně „hmatatelnými“ způsoby. Příkladem je náhled zákazníka. Zákazníci mohou ztratit důvěru v daný podnik, který dopustil, že k chybě došlo. Organizace zabývající se vývojem software musí následně utratit velké množství peněz, aby získaly konkrétní informace o zákazníkovi, najít problémy a opravit příčinu selhání. Jakýkoliv další čas programátora strávený na opravě chyb představuje náklady pro firmu. Každé firma, kde je vyvíjen jakýkoliv software, by si měla položit otázku, jak drahé je testování v porovnání se siutací, kdy se netestuje (včetně nehmotné nákladů i zjevnějších přímých nákladů). [36]

Důležitou věcí je také zvážení relativního rizika spojeného se selháním v závislosti na typu projektu, na kterém se pracuje. Kvalita je mnohem důležitější například pro bezpečnostní či letecký software, než pro videohry. Ve skutečnosti může software kritický z hlediska bezpečnosti utratit za testování třikrát až pětkrát více než všechny ostatní kroky softwarového inženýrství dohromady.

Aby se minimalizovaly náklady spojené s testováním a se selháním softwaru, musí být cílem testování odhalit co nejvíce defektů s co nejmenším množstvím testování. Jinými slovy, je třeba napsat testovací scénáře, u kterých je vysoká pravděpodobnost odhalení závad. Je téměř nemožné otestovat všechny možné kombinace vstupů a výstupů systému; je prostě příliš mnoho permutací a kombinací. Testeři musí zvážit ekonomiku testování a snažit se psát testovací scénáře, jež odhalí co nejvíce chyb v co nejmenším počtu testovacích případů. [36]

## 2.4 Technologie využité pro vývoj webové aplikace

### 2.4.1 Uživatelské rozhraní

#### 2.4.1.1 EJS

Webové aplikace jsou obvykle založené na souborech s koncovkou „.html“. Takzvaným „view engine“ je tedy HTML. V Node.js aplikaci je možné místo HTML využívat EJS. Tento typ souboru má výhodu v možnosti psát Javascript kód přímo mezi HTML kód. Díky této funkcionalitě je v mnoha případech možné kód generovat za pomoci loopu, čímž se soubory značně zkrátí. Další užitečnou funkcionalitou využívanou i v této práci je funkce „include“, díky níž je možné části kódu, které se opakují ve více souborech (například navigační menu, header, footer apod.) z těchto souborů odebrat, vložit do separátního souboru a za pomoci této funkce pouze do zbylých souborů naimportovat. Každá změna se tak projeví na všech stránkách aplikace stejně a není třeba je dělat zvlášť. [21]

#### 2.4.1.2 CSS

CSS je zkratka pro anglický termín „Cascading Style Sheets“, v překladu tedy „kaskádové styly“. Jedná se o jazyk používaný k nastylování dokumentu psaného ve značkovacím jazyce, tedy HTML. HTML kód vytvoří strukturu stránky, která však nemá žádný vzhled. CSS je navrženo tak, aby umožňovalo oddělení prezentace a obsahu, včetně rozvržení, barev a písem. Toto oddělení může zlepšit dostupnost obsahu, poskytnout větší flexibilitu a kontrolu při specifikaci prezentačních charakteristik, umožnit více webovým stránkám sdílet formátování zadáním příslušného CSS v samostatném souboru .css, což snižuje složitost a opakování strukturálního obsahu, a povolit ukládání souboru .css do mezipaměti, aby se zvýšila rychlost načítání stránky mezi stránkami, které sdílejí soubor, a jeho formátování. Oddělení formátování a obsahu také umožňuje prezentovat stejnou značkovací stránku v různých stylech pro různé metody vykreslování, například na obrazovce, v tisku, hlasem či pomocí Braillova písma. CSS má také pravidla pro alternativní formátování, pokud je obsah přístupný na mobilním zařízení. [20]

### 2.4.2 Javascript

Javascript je v posledních letech populární programovací jazyk. Využívá se především k při tvorbě webových aplikací. Javascript běží na straně klienta, nikoliv na serveru. Ačkoliv například knihovna Node.js již serverový Javascript podporuje. [25]

Vznik Javascriptu má své počátky spojené se společností Sun, ve které v roce 1992 vznikl jazyk Java. V Sun viděli příležitost v použití webu, avšak v té době zvládaly prohlížeče zobrazovat pouze statický obsah bez jakýchkoliv interaktivních prvků. Pro tyto potřeby začaly vznikat JavaApplety běžící na straně klienta. Ty umožňovaly vkládat na web Java aplikace. [26]

Netscape přistupuje k této problematice jinak. Během velmi krátkých 3 týdnů vzniká jazyk LiveScript díky spojení syntaktických pravidel Javy, funkcionálního paradigma Scheme a prototypového programovacího jazyka Self. K přejmenování z LiveScript na JavaScript došlo pouze z marketingových důvodů. Java a JavaScript nemají kromě částečných syntaktických podobností nic společného. [28]

#### 2.4.2.1 jQuery

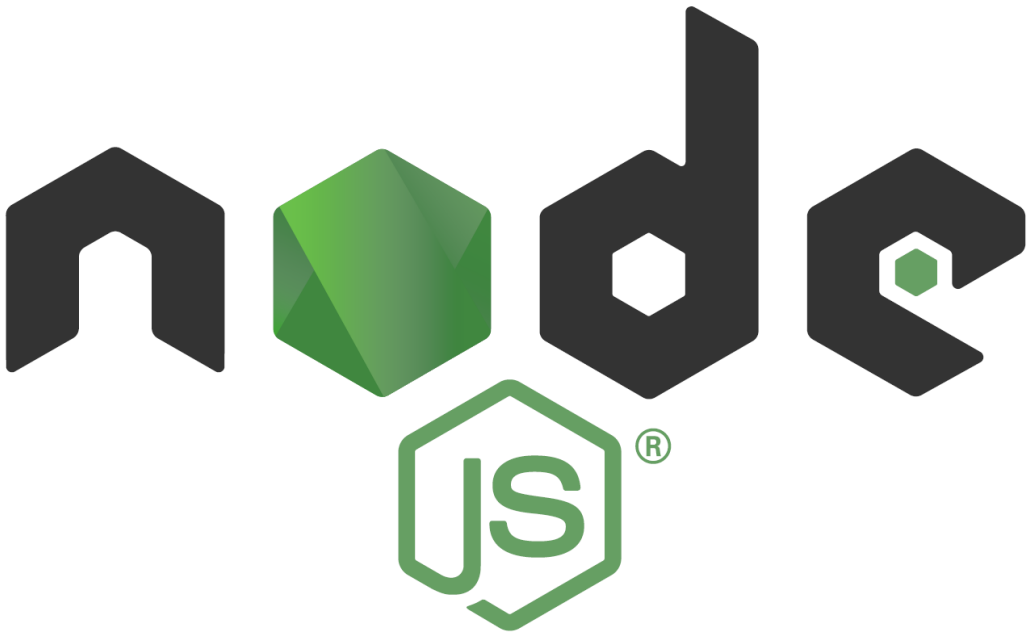
jQuery je JavaScriptová knihovna navržená tak, aby zjednodušila procházení a manipulaci s HTML DOMem, stejně jako zpracování událostí, CSS animace i Ajax. Jedná se o bezplatný open-source software využívající licenci MIT. Od května 2019 používá jQuery 73 % z 10 milionů nejoblíbenějších webových stránek. Webová analýza ukazuje, že je to s velkým náskokem nejrozšířenější JavaScriptová knihovna, která má alespoň 3 až 4krát větší využití než jakákoli jiná JavaScriptová knihovna. [23]

#### 2.4.2.2 Node.js

Node.js je open-source, multiplatformní, back-endové běhové prostředí JavaScriptu, které běží na enginu V8 a spouští kód JavaScript mimo webový prohlížeč. Node.js umožňuje vývojářům používat JavaScript k psaní nástrojů

příkazového řádku a pro skriptování na straně serveru – spouštění skriptů na straně serveru k vytvoření dynamického obsahu webové stránky před odesláním stránky do webového prohlížeče uživatele. V důsledku toho Node.js představuje paradigma „JavaScript všude“, které sjednocuje vývoj webových aplikací kolem jediného programovacího jazyka, spíše než různých jazyků pro skripty na straně serveru a na straně klienta. [29]

Node.js má událostmi řízenou architekturu schopnou asynchronního I/O. Cílem těchto návrhů je optimalizovat propustnost a škálovatelnost ve webových aplikacích s mnoha vstupně/výstupními operacemi, stejně jako pro webové aplikace v reálném čase (např. komunikační programy v reálném čase a hry v prohlížeči). [29]



**Obr. 3 Logo Node.js**
Zdroj: https://nodejs.org/en/about/resources/

#### 2.4.2.3 Express.js

Express.js, nebo jednoduše Express, je back-end webový aplikační framework pro Node.js, vydaný jako bezplatný open-source software pod licencí MIT. Je určen pro vytváření webových aplikací a API. Je nazýván de facto

standardním serverovým frameworkem pro Node.js. Hlavními výhodami Express.js ja například možnost robustního routování, vysoký výkon, pomoc pro HTTP jako je přesměrovávání mezi stránkami či cachování. [22]

```
1   const express = require('express');
2
3   const app = express();
4
5   app.get('/', (req, res)=>{
6       res.send('Hi, your request has been received');
7   });
8
9   app.listen(3000, ()=>{
10      console.log('Listening at http://localhost:23000');
11  });
```

**Obr. 4 Základní implementace Express.js**
Zdroj: Vlastní zpracování

#### 2.4.2.4 AJAX

AJAX je anglická zkratka „Asynchronous JavaScript And XML“, tedy „asynchronní Javascript a XML“. Nejedná se o programovací jazyk, ale pouze o techniku, která slouží k přístupu k webovým serverům z webové stránky. [27]

Díky AJAXu je možné na klientovi v prohlížeči načíst veškerá data například zadaná uživatelem, odeslat je v podobě objektu na server a tam s nimi dále pracovat (např. uložit do databáze). Obecně tedy AJAX zastřešuje technologie, které umožňují komunikaci prohlížeče a serveru bez nutnosti obnovení aktuální stránky. [27]

#### 2.4.2.5 XMLHttpRequest

XMLHttpRequest (XHR) je API ve formě objektu, jehož metody přenášejí data mezi webovým prohlížečem a webovým serverem. Objekt je poskytován JavaScriptovým prostředím prohlížeče. XHR je například hojně využíváno u Ajaxu.

Základním konceptem návrhu Ajaxu je zejména získávání dat z XHR za účelem neustálého upravování načtené webové stránky. Navzdory názvu lze XHR používat s jinými protokoly než HTTP a data mohou být ve formě nejen XML, ale i JSON, HTML nebo prostého textu. [24]

### 2.4.3 Databáze MongoDB

MongoDB patří mezi NoSQL databáze. Nelze tedy nad nimi provádět operace v SQL jazyce sloužící pro zpracování dat v relačních databázích. Data jsou do MongoDB ukládána v podobě objektu velmi podobnému JSONu (pole objektů). V MongoDB je možné vytvořit mnoho databází, které se dále dělí na kolekce, které se dále dělí na dokumenty. Kolekce jsou v tomto případě soubory dokumentů většinou se stejnou strukturou JSONu. Dokument je nejmenší částí, tedy jedním zápisem, jedním JSON souborem. [28]

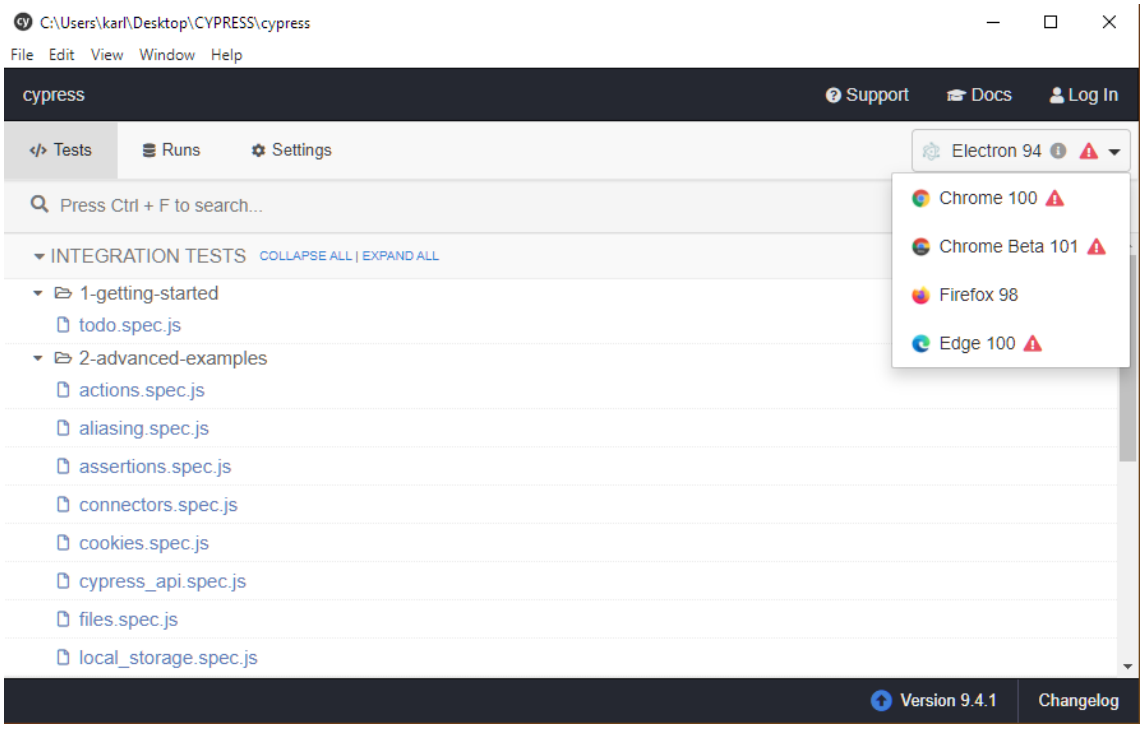
### 2.4.4 Cypress automatické testy

Cypress je testovací framework založený na jazyce JavaScript. V rámci cypress je pak kromě JavaScriptu možné využít ještě TypeScript, avšak jiné jazyky nejsou podporovány. Jeho hlavní charakteristikou je běh v prohlížeč. Cypress nepodporuje všechny prohlížeče, avšak pouze omezené množství. Konkrétně se jedná o Google Chrome, Mozilla Firefox, Microsoft Edge, Electron a Brave. [31]

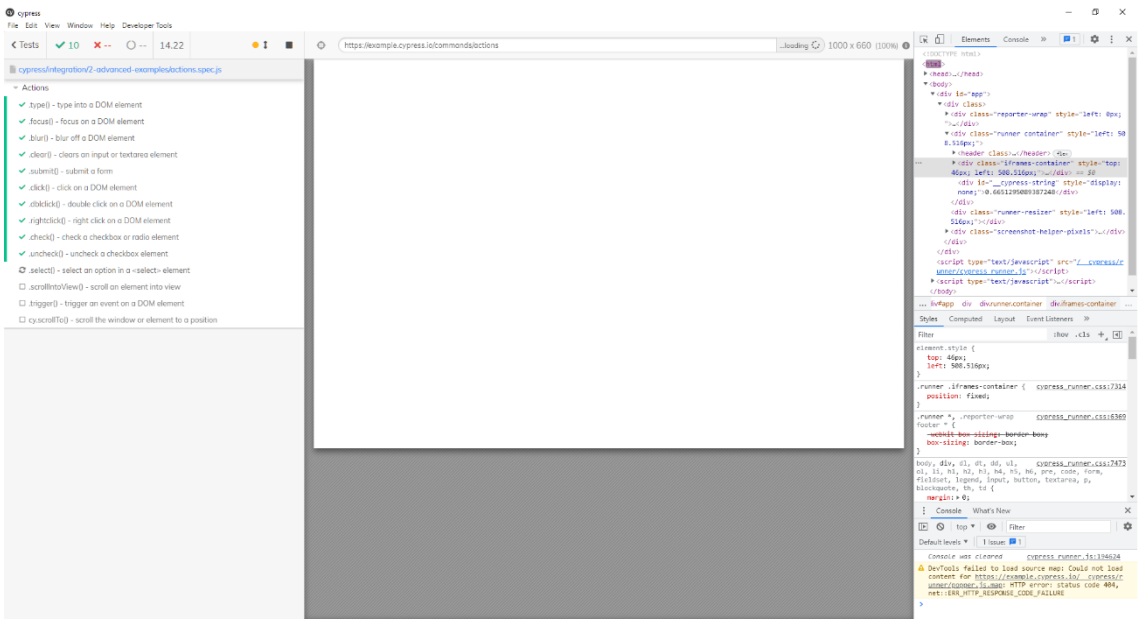
Cypress je velmi rychlý a je zcela nezávislý framework. Jedná se o bezplatný a lehce dostupný open-source nástroj. Cypress má v sobě implementovaných mnoho šikovných funkcí pro testování. Jednou z nich je automatické čekání na načtení DOMu, případně velmi lehce napsatelné čekání na konkrétní element DOMu po přesně daný čas. Dalšími automaticky integrovanými funkcemi cypressu je pořizování videa celého testu a vytváření screenshotů v případě chyby během testu. Screenshoty je pak možné pořizovat v jakémkoliv kroku testu za pomoci příkazu cy.screenshot();. [31]

Hlavními komponentami cypressu jsou anglické pojmy „Test Runner“ a „Dashboard“ ) . Dashboard lze chápat jako jakési hlavní uživatelské rozhraní pro

snadné spouštění testů. V rámci cypress dashboardu je kromě spouštění testů možné nastavit prohlížeč, ve kterém jsou testy spouštěny, je možné zde nastavit obecné atributy cypressu a testů apod. [31]




**Obr. 5 Cypress dashboard**
Zdroj: Vlastní zpracování



**Obr. 6 Cypress runner**

Zdroj: Vlastní zpracování

Druhou základní komponentou je „Test Runner“, což je okno, které se zobrazí po spuštění testu a test zde běží. V levé části okna jsou zobrazeny testovací scénáře, které jsou postupně načítány a následně označeny podle výsledku. Pokud test projde, je označen zeleným „tickem“, pokud neprojde, je označen červeným křížkem. Cypress měří i čas od spuštění a je možné zde test dále ovládat. Největší částí je pak sekce, kde probíhá samotný test. [32]

Cypress není možné spustit bez předchozí přípravy. Předpoklady pro napsání a spuštění testů v cypressu jsou následující: [32]

- mít nainstalovaný Node.js z důvodu, že cypress běží na node serveru.
- mít nainstalovaný cypress v projektu
- mít nainstalovaný editor pro úpravu kódu či nejlépe IDE

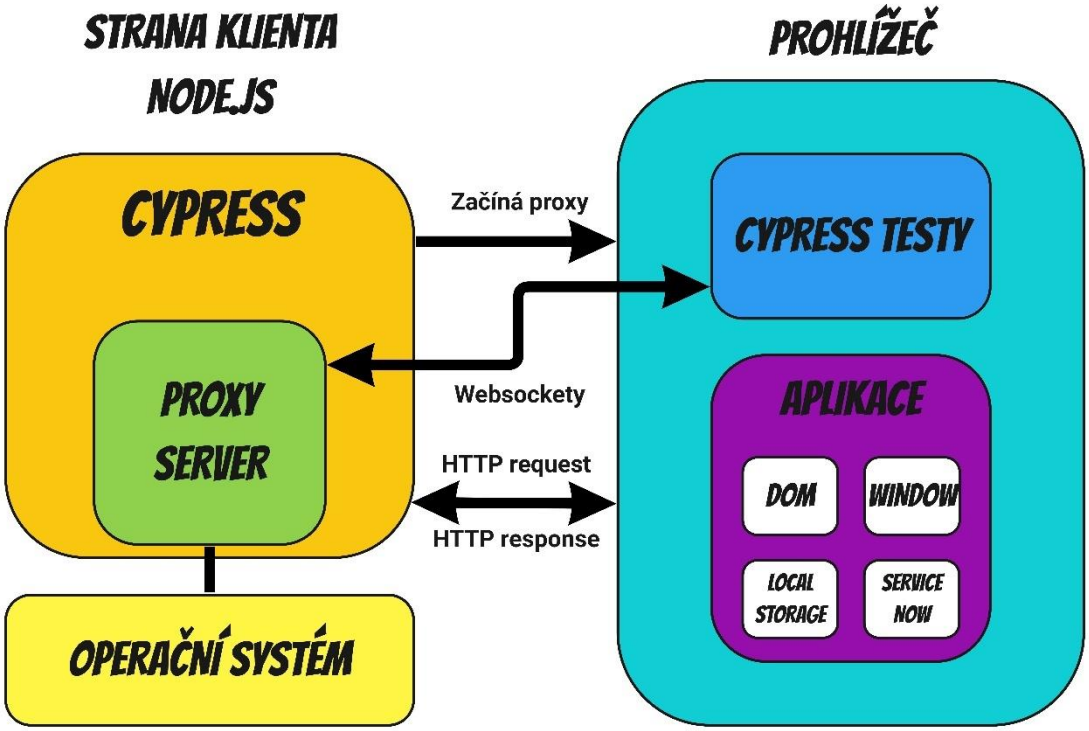
Většina testovacích nástrojů běží na serveru mimo prohlížeč a provádí příkazy po síti. Cypress na druhou stranu běží v prohlížeči, kde je také spuštěna aplikace. Tímto způsobem může přistupovat ke všem DOM elementům a ke všemu uvnitř prohlížeče. [32]

Node server běží za Cypressem na straně klienta. Node server a Cypress tedy vzájemně spolupracují. Vzhledem k tomu, že cypress má přístup k front-endu i back-endu, je odezva na aplikaci v reálném čase velmi dobrá. Cypress může provádět i úkony, které běží mimo prohlížeč. [30]

Cypress interaguje též se síťovou vrstvou a zachycuje příkazy čtením a změnou webového trafficu. Odesílá též HTTP requesty z node serveru a přijímá HTTP odezvy (response) od prohlížeče. Komunikace mezi node serverem a prohlížečem probíhá skrz WebSockety. Tato komunikace začíná po spuštění proxy serveru. [30]

Cypress ovládá všechny příkazy, které se spouštějí v prohlížeči i mimo něj. Vzhledem k tomu, že cypress je nainstalován lokálně, kooperuje též s operačním systémem zařízení za účelem nahrávání videí, pořizování snapshotů, přístupu k síťové vrstvě a provádění operací spojených se systémem souborů a složek.

Cypress má též přístup ke spoustě nástrojů a částí v prohlížeči jako je DOM, objekt window, local storage, network layer a DevTools. [30]




**Obr. 7 Princip cypressu**
Zdroj: https://lambdageeks.com/cypress-automation-cypress-architecture-install-cypress/

### 2.4.5 Modelovací standardy

Pro správné a názorné vymezení testovacího procesu a funkcionality testovací aplikace je možné využít mnoho modelovacích standardů jako jsou UML (Unified Modelling Language), BPMN (Business Process Model and Notation), BPEL (Business Process Execution Language) či například EPC (Event-driven Process Chains). Níže jsou uvedeny informace o prvních dvou zmíněných standardech.

#### 2.4.5.1 UML

Unified Modeling Language (UML) je univerzální, vývojový, modelovací jazyk v oblasti softwarového inženýrství, který má poskytovat standardní způsob vizualizace návrhu systému. [34]

Vytvoření UML bylo původně motivováno touhou standardizovat různorodé notační systémy a přístupy k návrhu softwaru. V roce 1997 byl UML přijat jako standard Object Management Group (OMG) a od té doby je touto organizací řízen. V roce 2005 byl UML také publikován Mezinárodní organizací pro standardizaci (ISO) jako schválený standard ISO. Od té doby byl standard pravidelně revidován, aby pokryl nejnovější revizi UML. V softwarovém inženýrství většina praktiků nepoužívá UML, ale místo toho vytváří neformální ručně kreslené diagramy; tyto diagramy však často obsahují prvky z UML. [34]

#### 2.4.5.2 BPMN

Business Process Model and Notation (BPMN) je grafické znázornění pro specifikaci obchodních procesů v modelu obchodních procesů. BPMN, původně vyvinutá iniciativou Business Process Management Initiative (BPMI), je spravována skupinou Object Management Group (OMG) od sloučení těchto dvou organizací v roce 2005. Verze BPMN 2.0 byla vydána v lednu 2011, kdy název byl upraven na Business Process Model and Notation, aby odrážel zavedení sémantiky provádění, která byla zavedena spolu se stávajícími prvky notace a diagramů. Přestože se jedná o specifikaci OMG, BPMN je také ratifikován jako ISO 19510. [35]

# 3 Praktická část

## 3.1 Proces optimalizace

Jak již bylo uvedeno v teoretické části, po optimalizaci by mělo být pozorovatelné poměrně velké množství zlepšení. Ne všechna uvedená zlepšení je možné pozorovat u všech procesů. Některá není možné aplikovat, případně je není možné snadno či vůbec jakýmkoliv způsobem měřit nebo identifikovat.

V rámci optimalizačního procesu bude dodržováno pořadí kroků určené v teoretické části práce, tedy identifikace, přehodnocení, implementace, automatizace a monitorování.

Prvním krokem pro správnou optimalizaci je třeba identifikovat místa v procesu, která nejsou zcela efektivní a tyto místa následně zlepšovat. Aby bylo možné tato místa lépe rozpoznat, bude v první řadě rozebrán celý stávající proces. Následně za pomoci SWOT analýzy určíme silné a slabé stránky, příležitosti a hrozby. Na základě SWOT analýzy budou pak provedena příslušná opatření (tj. fáze předhodnocení). Implementace a automatizace bude pro účely diplomové práce spíše teoretickou záležitostí. Proběhne sice kompletní přepracování stávající flow a výměna za flow novou, avšak zatím pouze v teoretické rovině, protože implementace do firemního prostředí se uskuteční až později. Ze zcela stejného důvodu nebude poslední fáze, tedy monitorování, již předmětem diplomové práce. Nebude možné měřit zlepšení v praxi a tím pádem zcela přesně určit naplnění zadaných cílů.

## 3.2 Popis vybraného pracovního procesu

Tato diplomová práce a aplikace s ní spojená je určena pro jeden konkrétní obsáhlý a časově náročný testovací proces. Jedná se o proces testování nově vytvořeného produktu. Účelem tohoto procesu není otestovaní přímo funkce software, který zákazník využívá, ale následujícího výčtu nastavení a funkcionalit:

- zalistování všech variant produktu v interních a externích systémech

- zobrazení variant produktu v e-shopu (webový košík i tzv. IPM košík, tj. košík na klientovi, tedy v aplikaci), jejich chování a nákup
- správné nastavení cen všech variant produktů
- automatické obnovení licence
- licenční operace na back-endu
- e-mailové notifikace
- zobrazení a nastavení produktu v uživatelském účtu
- chování licence v aplikaci

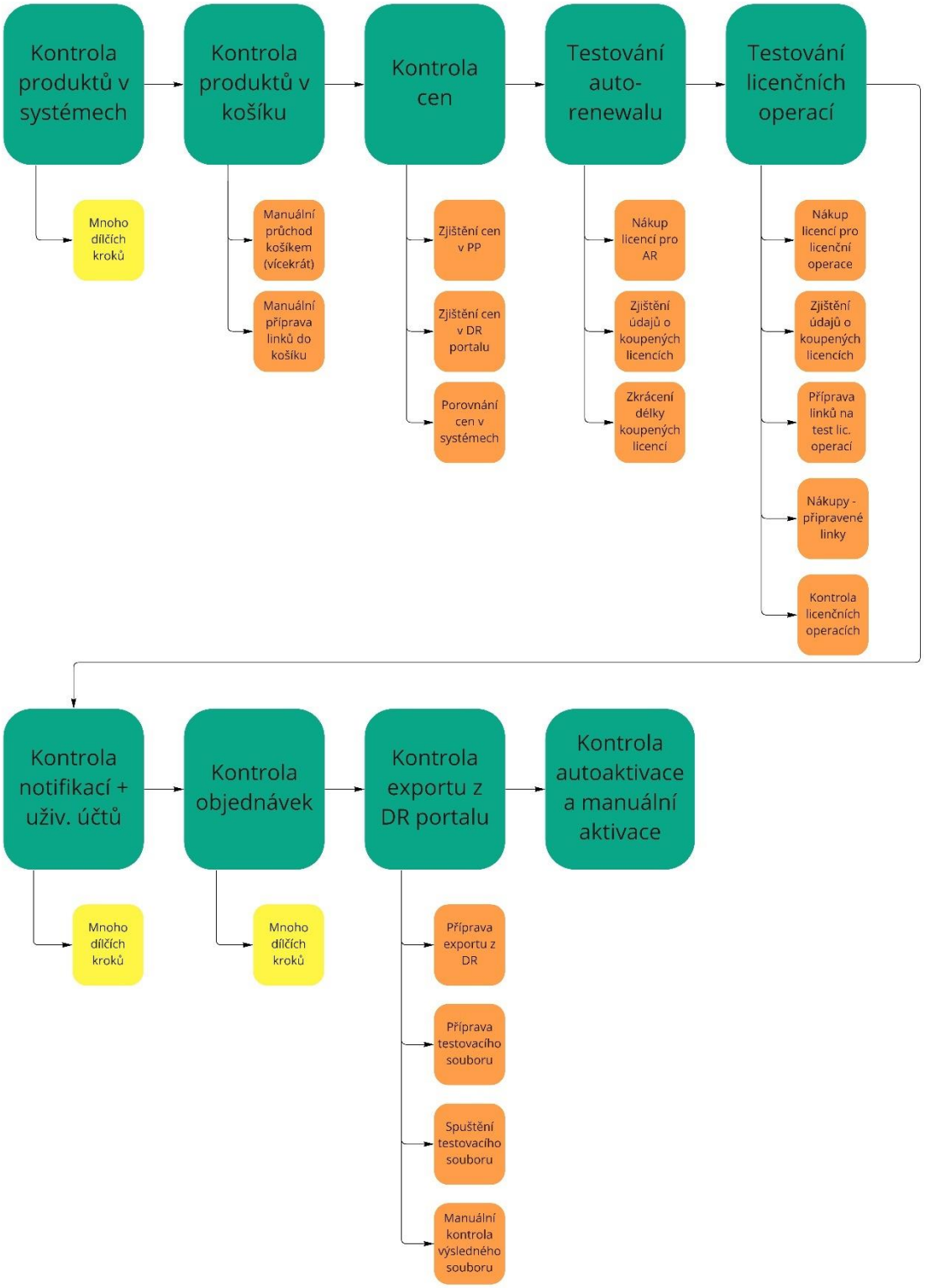
Výčet uvedený výše sice čítá pouze 7 bodů, nicméně každý z nich obsahuje až desítky dalších podbodů. Kompletní otestování je tedy časově náročný proces. Z toho důvodu je třeba jej zoptimalizovat, zkrátit, zlehčit.

Hlavním výstupem této práce je webová aplikace pokrývající celé výše zmíněné flow ulehčující všechny jeho kroky. Aplikace kopíruje kroky testovacího procesu a pomáhá je zrychlovat a zlehčovat. Součástí práce je i několik automatických testů.

## 3.3 Popis současného stavu procesu

Proces produktového testování je velmi obsáhlý proces, kdy je třeba na nově založený produkt nahlížet z mnoha stran. Doposud je celý proces založen na manuálním testování, což jej dělá značně pomalejším a náročnějším než by měl být po implementaci řešení z této diplomové práce. V současné době jsou testovací scénáře zaznamenány v jednoduchém checklistu a je třeba postupovat sekvenčně krok po kroku.

Schéma níže znázorňuje současnou flow celého procesu. Schéma obsahuje několik obsáhlejších kroků (označeno zeleně), které se dále dělí na menší celky (označeny oranžově). Všechny tyto celky znázorňují kroky celého procesu a platí, že doposud jsou prováděny sekvenčně.




**Obr. 8 Původní testovací flow**
Zdroj: Vlastní zpracování

**KONTROLA PRODUKTŮ V SYSTÉMECH**

První částí, jak již bylo zmíněno v předchozí části (viz. 3.2.), je kontrola zalistování všech variant produktů v interních i externích systémech. Jedná se o celkem 4 systémy, kde je nutné kontrolovat rozličné konfigurace produktů.

**KONTROLA PRODUKTŮ V KOŠÍKU**

Druhou částí, která je značně časově náročná a v současné době ne zcela efektivní, je protestování produktu z pohledu uživatele v košíku (t.j. e-shop). Je třeba otestovat nákup produktu přes rozličné platební metody, zobrazení produktu v košíku (IPM, tedy košík běžící v aplikaci i webový košík v prohlížeči). V ideálním případě je třeba otestovat všechny varianty produktu ve všech regionech, které jsou pro daný produkt definovány. Z tohoto vyplývá, že v ideálním případě je třeba otestovat opravdu velké množství kombinací, což není při manuálním testování možné v přijatelném časovém úseku.

**KONTROLA CEN**

Třetím testem je otestování cen. V tomto případě se jedná o porovnání cen všech variant produktů (ve všech měnách a regionech) ve dvou systémech a následně i v košíku. Pro tuto část platí stejné tvrzení jako pro část předchozí – jedná se o zdlouhavý a neefektivní proces. V tomto případě má manuální testování další slabinu, a tou je chybovost lidského faktoru. Je velmi snadné udělat chybu při porovnávání cen.

**KONTROLA AUTO-RENEWALU**

Tento test by měl kontrolovat automatickou obnovu licence uživatele. Není to pravidlem, ale většina produktů má ve výchozím nastavení nastavenou tuto automatickou obnovu neboli autorenewal. Pokud uživateli licence vyprší, dojde automaticky k jejímu obnovení o jeden rok. Tuto operaci lze provést, avšak pouze manuálně. Po nákupu první licence je možné tuto licenci v licenčních systémech „zkrátit“, tedy zkrátit čas než licence vyexpiruje a následně zkontrolovat její obnovení.

**KONTROLA LICENČNÍCH OPERACÍ**

Pátým testem jsou testy licenčních operací. Pro tyto operace jsou opět, stejně jako ve druhém kroku, potřeba nákupy, v ideálním případě všech variant produktu. Následuje zjišťování informací o těchto licencích a příprava url adres pro další testování. V tomto případě se nejedná o nikterak náročný proces, nicméně prvotní nákup produktů může být opět zdlouhavý.

**KONTROLA NOTIFIKACÍ a UŽIVATELSKÝCH ÚČTŮ**

Následuje kontrola notifikací. Uživateli během životního cyklu licence je zasíláno poměrně velké množství notifikací, konkrétně jejich e-mailová podoba. Účelem tohoto testu je odhalit chyby v základních typech e-mailů, kterými je potvrzovací e-mail o dokončeném nákupu, e-mail s možností verifikace na back-endu vytvořeném uživatelském účtu a poté i tzv. autorenewal confirmation e-mail (e-mail zaslaný při expiraci předchozí licence a jejím automatickém obnovení). Notifikace jsou testovány kvůli jejich designu, nicméně hlavním důvodem je kontrola licenčních údajů vygenerovaných na back-endu a zaslaných uživateli a též funkcionalit v notifikacích jako například tlačítka pro stažení nakoupených produktů apod.

Jak již bylo zmíněno, uživateli je při nákupu automaticky vytvořen účet pro správu licencí. S tím je spojeno též mnoho dalších dílčích testů, které je třeba provést.

**KONTROLA OBJEDNÁVEK**

Tento test je opět čistě manuálního charakteru. Licence nakoupené ve druhém kroku, tedy při kontrole košíku, je možné využít v tomto testu. Nově vytvořené licence se propisují ve velkém množství systémů – systémy poskytovatelů košíku, licenční systémy a mnoho dalších. V těchto systémech je třeba objednávky zkontrolovat, že veškeré jejich atributy jsou v pořádku. Atributů, které je třeba kontrolovat, je velké množství. Z tohoto důvodu je zde nebudu všechny uvádět.

## KONTROLA EXPORTU Z DR PORTÁLU

Produkty zalistované v systému poskytovatele košíku Digital River nejsou kontrolovány podrobně přímo v systému, ale v exportu z tohoto systému. Atributy produktů se mohou lišit pro různé regiony. Každý region je v exportu na separátním listu v excelovském souboru, proto není jednoduché test provést bez chyb.

Z tohoto důvodu jsem vytvořil již v minulosti soubor s koncovkou .xlsm, tedy soubor podporující makra v excelu. Za pomoci jazyka Visual Basic integrovaného do souboru Microsoft Office Excel je možné kontrolovat export za pomoci několika málo kliknutí myši. Soubor vytvoří ze všech sešitů jeden velký sešit odlišený dle regionů. Sloupce s atributy založenými na HTML kódu, jsou „přeloženy“ a přetvořeny do podoby html stránek pro snadnější kontrolu. Tento test je doposud zatím jediný v celém procesu, kde jsou viditelné náznaky automatického testování. Ačkoliv je soubor funkční, není moc efektivní zpracovávat větší množství dat z důvodu časové náročnosti, a proto si i tento krok zaslouží optimalizaci, respektive automatizaci za využití jiných technologií

## KONTROLA AUTOAKTIVACE A MANUÁLNÍ AKTIVACE

Dalším testem je kontrola aktivace licence v produktu. Uživateli, který si zakoupí licenci, musí být umožněno licenci aktivovat v aplikaci. Pokud by tato operace nebyla funkční, zákazník nemůže licenci využít. Z tohoto důvodu je třeba manuální aktivaci v produktu otestovat z uživatelského pohledu. Jedná se především o licence zakoupené přes webový košík v prohlížeči. Licence zakoupené přímo v programu, by se ve většině případů, měly samy aktivovat. To je další z testů, které je třeba provést.

## 3.4 SWOT analýza stávajícího stavu

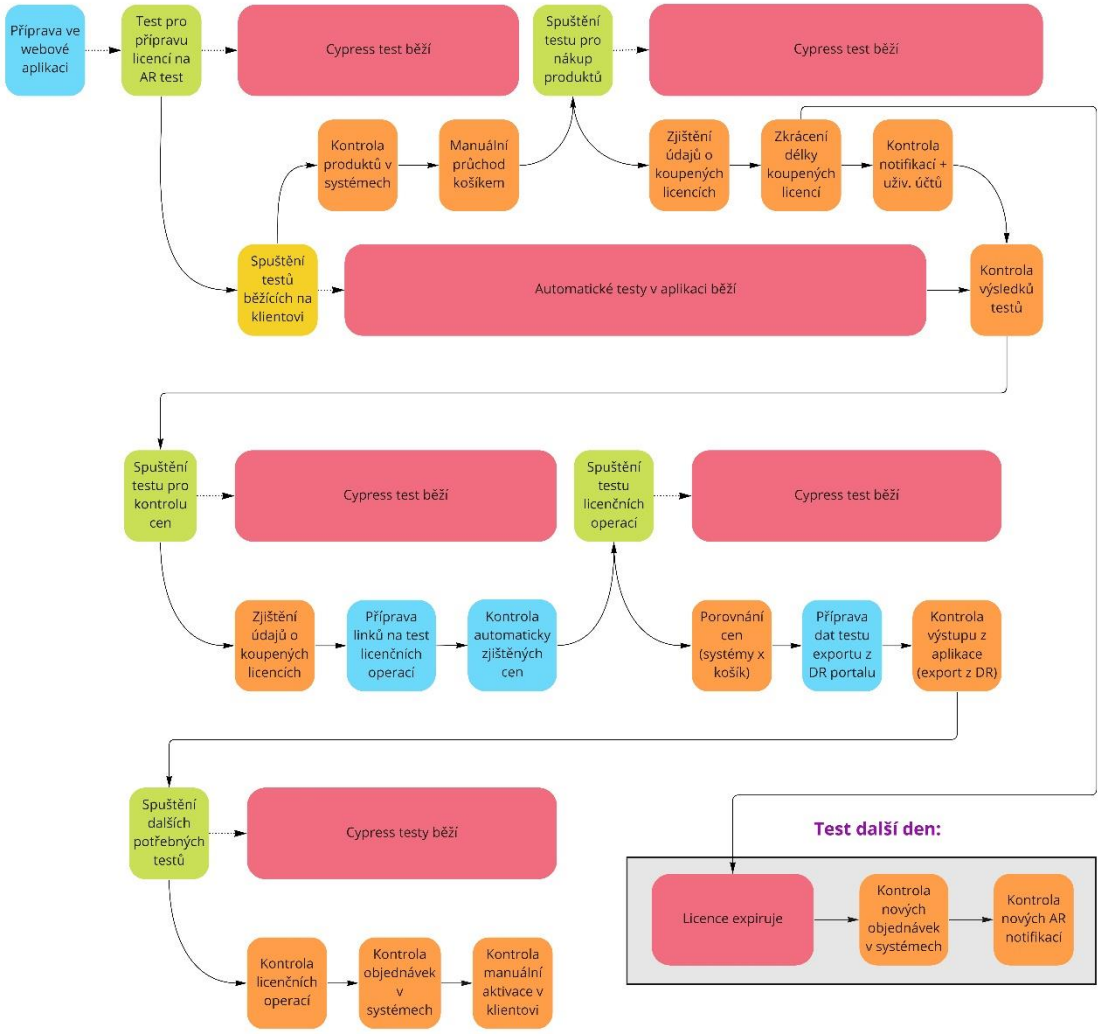
Pro správnou optimalizaci stávajícího stavu procesu je třeba provést jeho analýzu a vyhodnotit různé aspekty tohoto procesu, aby tyto aspekty mohly být zachovány či naopak nahrazeny lepším řešením. Pro takto koncipovanou analýzu se nabízí využití SWOT analýzy.

| | **POZITIVNÍ** | **NEGATIVNÍ** |
|---|---|---|
| **VNITŘNÍ PŮVOD** | **SILNÉ STRÁNKY**<br>• Široké pokrytí procesu testy zaměřených na uživatelský pohled.<br>• Kvalitně vytvořený checklist s testovacími scénáři. | **SLABÉ STRÁNKY**<br>• Zdlouhavý proces<br>• Proces je bez automatizace<br>• Sekvenčnost dílčích testů<br>• Malé množství testovaných dat u některých testů (testování lokalizačních odlišností). |
| **VNĚJŠÍ PŮVOD** | **PŘÍLEŽITOSTI**<br>• Testování krajových zvláštností (nesprávné chování zákazníka).<br>• Větší zaměření na UX (user experience) | **HROZBY**<br>• Nemožnost otestovat produkt ve všech jeho podobách z důvodu přístupů, práv apod. |

**Tabulka 1 SWOT analýza původní flow**
Zdroj: Vlastní zpracování

## 3.5 Aplikace optimalizačních technik do procesu

Následující schéma zobrazuje nově navržené flow testovacího procesu po implementaci testovací aplikace a automatických testů do procesu a po celkové optimalizaci flow.




**Obr. 9 Nová optimalizovaná flow**
Zdroj: Vlastní zpracování

Hlavní optimalizační technikou je automatizace některých procesů. Velké množství dílčích testů však zautomatizovat nejde. Pro tyto procesy byly navrženy a vytvořeny nástroje, které by měly manuálních testy usnadnit a urychlit.

Co je třeba říci k nově vytvořené flow je to, že není zcela dodrženo původní pořadí úkonů, ačkoli jednotlivé kroky v aplikaci kopírují původní flow. Důvodem je

optimalizace celé flow z časového hlediska a nejlepší rozvržení spouštění automatických testů. Díky změně pořadí jednotlivých kroků bylo možné ze zcela sekvenčního procesu vytvořit proces běžící ve 2 až 3 vláknech, což vede k urychlení procesu. Automatické testy stačí na začátku nastavit a spustit a poté je možné se věnovat jiným činnostem. Na tomto základu je nový přístup z velké části založen.

**KONTROLA PRODUKTŮ V SYSTÉMECH**

V této sekci nedošlo téměř k žádnému zlepšení. Jedná se o jednu z částí procesu, kde je třeba manuální testování z důvodu, že každý produkt může být nastaven jinak a nelze zcela generalizovat jejich nastavení nebo klasifikovat produkty do skupin, kde by se parametry nelišily.

**KONTROLA PRODUKTŮ V KOŠÍKU**

V této sekci došlo k nejmarkantnějšímu posunu. Všechny typy košíku napříč poskytovali košíku, regiony i produkty je možné testovat za pomoci automatických testů. Díky API interního systému Pricing Portal je velmi snadné vygenerovat všechny potřebné košíkové linky a nad nimi pak provádět testy. Ve starém procesu bylo nutné všechny link generovat manuálně a pak procházet košíkovou flow manuálně link po linku. Současný přístup by měl ušetřit mnoho času. Automatické testy jsou též schopny protestovat všechny potřebné linky, ne pouze linky vybrané namátkou testerem jako tomu bývalo. V rámci procesu zůstal zachovaný pouze jeden manuální test, a to jeden průchod košíkem zakončený nákupem z důvodu, že by test nezachytil nějaké designové chyby či chování, které není třeba testovat pravidelně, proto není v testech podchyceno a může se jednat o tzv. „corner case“, tedy okrajovou zvláštnost.

**KONTROLA CEN**

Dalším testem, kde došlo ke značnému pokroku, je testování cen. V původním testovacím procesu bylo nutné ceny kontrolovat manuálně (porovnat ceny v pricing portalu a Digital River portalu se zadáním v excel souboru), což přinášelo velkou chybovost nehledě na časovou náročnost v případě většího

množství kontrolovaných produktů. Manuální testování cen v rámci nového procesu téměř odpadá. V novém testovacím procesu testuje aplikace ceny sama. Aplikace vyšle XMLHttpRequest na API obou zmíněných systémů, výsledky uloží do databáze a uživateli zobrazí v přehledných tabulkách včetně výsledku, zda ceny odpovídají. Tento test je možné provést nejen na úrovni aplikace, ale i na úrovní automatických cypress testů fungujících na stejném principu jako v aplikaci, avšak jsou ještě rozšířeny o testování cen přímo v košících, což v aplikaci není možné.

**KONTROLA AUTO-RENEWALU**

Tento test zůstává z větší části i nadále manuálním ve velmi podobné formě jako byl test původní. Důvodem je především citlivost informací o licenci a též fakt, že úprava licence není nikterak náročnou ani zdlouhavou činností. Usnadněním je snad pouze provedení prvních nákupů za pomoci automatického testu.

**KONTROLA LICENČNÍCH OPERACÍ**

Kontrola licenčních operací prošla určitými změnami. Při testu jedné operace jsou za potřeba dva nákupy, mezi nimiž je potřeba příprava linku pro druhý nákup. Oba nákupy je možné provést automaticky za pomoci napsaného cypress testu. Pro přípravu linku je stále třeba manuální činnost, nicméně aplikace obsahuje nástroj pro usnadnění. Parametry přidávané do linků se liší v závislosti na mnoha faktorech. Nástroj v aplikaci sám rozpozná, které atributy přidat. Pomáhá též s šifrováním linků a uživatele přesměruje do systémů, které jsou pro přípravu potřeba.

**KONTROLA NOTIFIKACÍ a UŽIVATELSKÝCH ÚČTŮ**

Toto je sekce testů, jež zůstává obsahem téměř nezměněná. Kontrola designu notifikací a jejich atributů je věc, kterou lépe zkontroluje lidské oko. Notifikací není při produktovém testování nikterak velké množství, tudíž by spuštění nástroje pro automatickou kontrolu zabralo poměrně stejně velké množství času a tudíž se tento přístup nevyplatí.

Jediným vylepšením procesu z pohledu designu notifikace, především její produktové části, která se liší pro každý produkt, je využití nástroje Litmus. Po

přeposlání notifikace na e-mailovou adresu vygenerovanou Litmusem, je e-mail zobrazen v mnoha různých platformách (Gmail, Microsoft Outlook, Mozilla Thunderbird a mnoho dalších).

**KONTROLA OBJEDNÁVEK**

Jedná se o testy, které z důvodu citlivosti dat není dobré automatizovat. Proto v této sekci nedošlo téměř k žádné změně kromě naprogramovaného nástroje v rámci aplikace, který by měl čistě manuální proces lehce urychlit.

**KONTROLA EXPORTU Z DR PORTÁLU**

Ačkoli nástroj na kontrolu exportu z externího systému DR portálu, který se nejvíce podobá automatickému nástroji, je již v procesu implementován, dojde k jeho nahrazení nástrojem lepším a rychlejším. V původním nástroji je třeba interakce uživatele a excelovského souboru pomocí tlačítek, která poměrně pomalu vykonávají dílčí kroky. Tato interakce by měla v novém nástroji zmizet. Uživateli pouze stačí nahrát vyexportovaný soubor do aplikace, která se postará o zbytek sama a značně rychleji. Uživateli pak stačí výstupní data zobrazit či stáhnout k další kontrole.

**KONTROLA AUTOAKTIVACE A MANUÁLNÍ AKTIVACE**

V rámci tohoto testu dojde k zapojení nových testů naprogramovaných v Seleniu kolegou z testerského týmu. Testy jsou již funkční, avšak nebyly zatím správně integrovány do celé flow produktového testování. Testy běží přímo v aplikaci a umí zjistit, zda autoaktivace proběhla správně.

Druhá část tohoto testu zůstává manuální. Je v ní třeba instalace potřebné aplikace, vložení licenčního klíče a následná kontrola, že vše funguje, jak má.

## 3.6 Uživatelské rozhraní

Webová aplikace slouží pro testovací účely v rámci firmy Avast, kde je oficiálním jazykem používaným pro oficiální komunikaci anglický jazyk, tudíž i tato aplikace je celé v anglickém jazyce. Jazykové mutace zde nejsou podporovány.

Prvním důležitým krokem při návrhu uživatelského rozhraní je rozvržení jednotlivých podstránek obsažených v menu. Testovací aplikace se skládá z následujícíc dílčích podstránek:

- Registrační stránka (registration page)
- Přihlašovací stránka (login page)
- Úvodní stránka (basic info page)
- Stránka s automatickými testy (automated tests page)
- Stránka pro testování zalistování v systémech (system listing page)
- Stránka pro testování košíku (t.j. e-shop) (cast testing page)
- Stránka pro testování cen (price testing page)
- Stránka pro testování licenčních operací (license testing page)
- Stránka pro testování e-mailů (e-mails testing page)
- Stránka pro testování objednávek (orders testing page)
- Stránka pro testování exportu z externího systému (export from DR page)
- Stránka s pomocnými funkcionalitami a nástroji (auxiliary tools page)

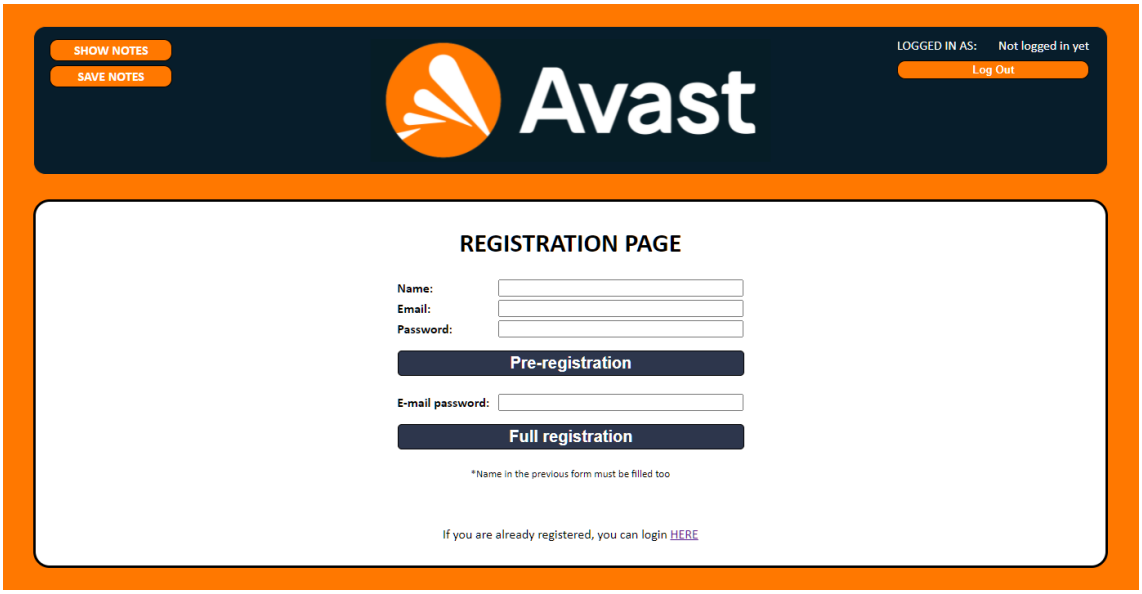
Celá webová aplikace má stejný barevný design platící pro všechny podstránky, avšak ne zcela stejnou strukturu. Struktura rozložení komponent je dvojí. První z nich se ve výčtu podstránek uvedeném výše týká prvních tří. To je design bez postranního panelu (menu). Ostatní podstránky obsahují navigační element.

Barevná struktura vychází ze základních barev typických pro firmu Avast. Dominantní barvou je oranžová (hexadecimálně zapsáno: #FF7800), která je využita i ve firemním logu umístěnému v hlavičce aplikace. Dalšími barvami jsou šedá, fialová a bílá, které jsou pro design aplikací, notifikací a dalších prvků spojených s Avastem velmi využívané.

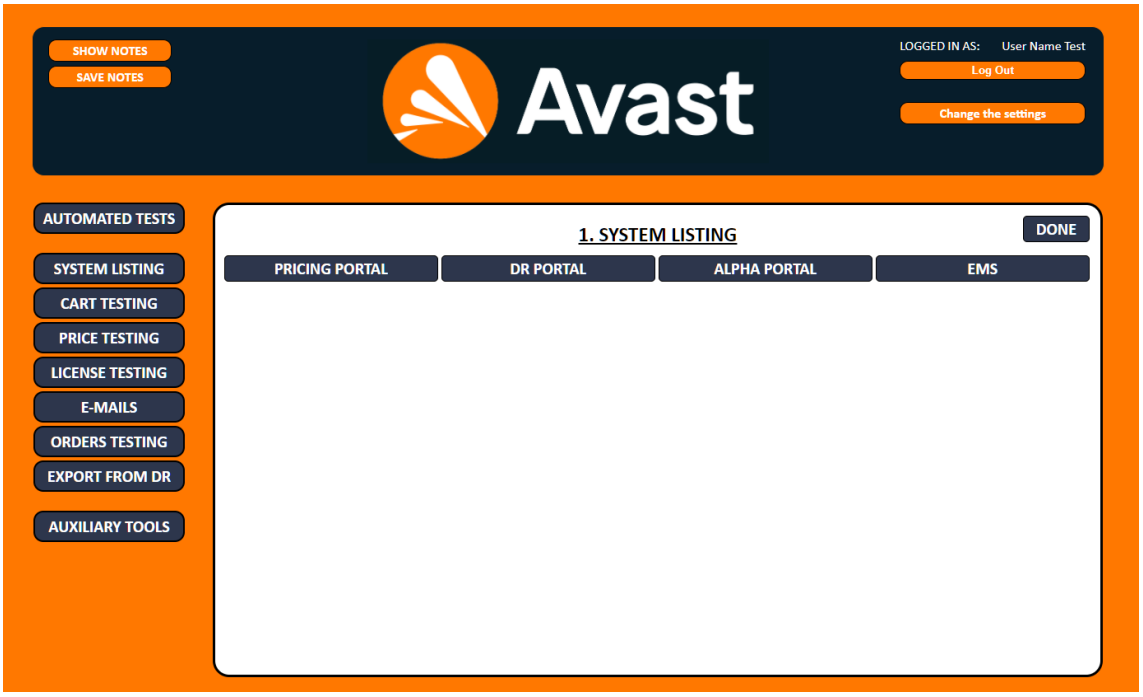
Vzhledem k tomu, že se jedná o webovou aplikaci určenou pro využívání pouze na počítačích (nikoliv na mobilních zařízeních), tak její design není zcela responzivní. Pro správnou funkčnost aplikace není tento typ designu stěžejní. Z pohledu kodérů, kteří tvoří webové stránky, by se následující přístup mohl zdát nesmyslný, nicméně v tomto případě je záměrný.

Každá stránka v obou designových variantách obsahuje hlavičku, která je dále složena z loga firmy, tlačítka pro zobrazení poznámek, které je možné uložit dalším tlačítkem, popř. jsou každých 10 vteřin ukládány automaticky. Tyto poznámky uživatele jsou dostupné napříč celou aplikací. Hlavička obsahuje i údaje o přihlášení/nepřihlášení uživatele.


**Obr. 10 Funkce přidávání poznámek**
Zdroj: Vlastní zpracování

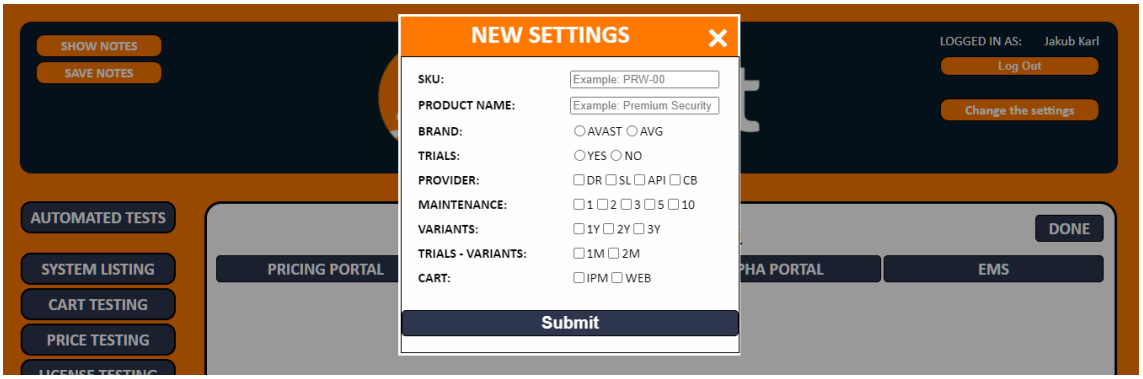



**Obr. 11 Registrační stránka aplikace**
Zdroj: Vlastní zpracování



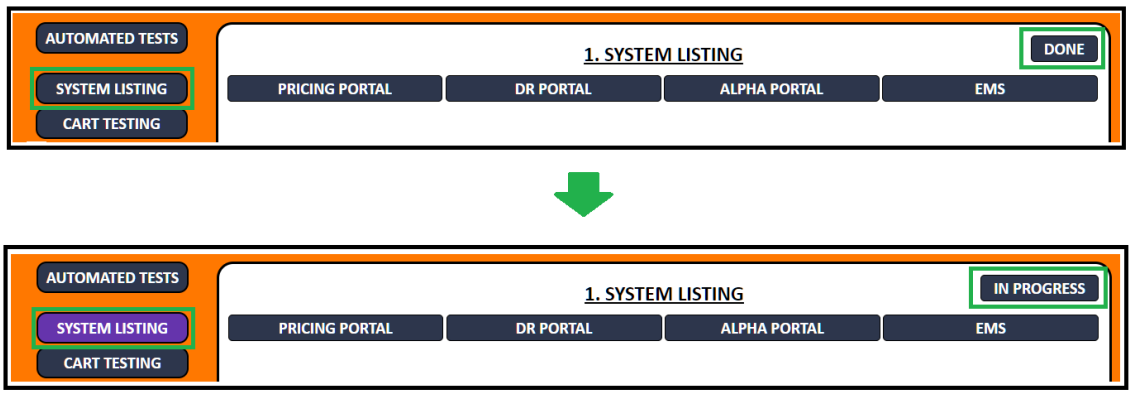

**Obr. 12 System listing stránka**
Zdroj: Vlastní zpracování

Jak je z obrázků patrné, hlavička se pro 1. a 2. typ designu liší v tlačítku „Change Settings“. Důvodem pro jeho absenci v první designové variantě je ten, že uživatel provádí základní nastavení až na úvodní stránce (t.j. basic info page), tudíž jeho změna nedává do tohoto kroku smysl. Po kliknutí na toto tlačítko se zobrazí modální okno s aktuálním nastavením, které je možné změnit a při potvrzení dojde ke změně všech potřebných dat v databázi.




**Obr. 13 Změna nastavení**
Zdroj: Vlastní zpracování

Další rozdílnou funkcionalitou pro obě skupiny podstránek je tlačítko „DONE“ umístěné na hlavní pracovní ploše aplikace v pravém horním rohu. Díky těmto tlačítkům je možné označit dokončené kroky testovacího procesu, které přesně odpovídají jednotlivým sekcím navigačního menu. Tato funkcionalita slouží k lepšímu zorientovaní se v celé workflow.




**Obr. 14 Označení otestované sekce**
Zdroj: Vlastní zpracování

## 3.7 Struktura webové aplikace

### 3.7.1 Registration page

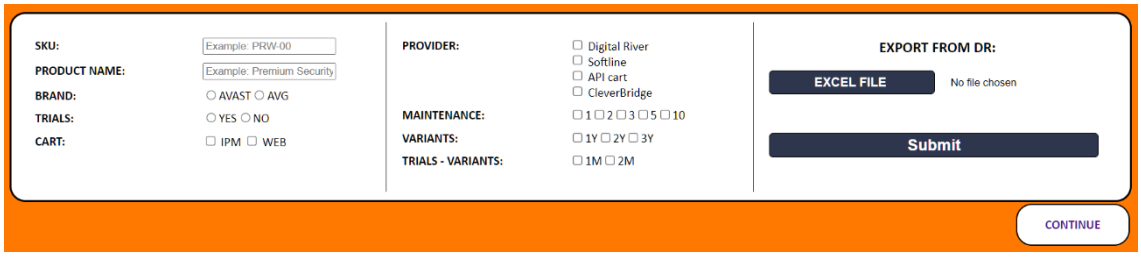
Tato stránka má velmi jednoduchou struktur. Obsahuje pouze registrační formulář složený ze dvou částí. První částí jsou samotné údaje o uživateli, tj. jméno, e-mailová adresa a heslo, po jejímž vyplnění a potvrzení je na e-mail zadaný uživatelem odeslána notifikace s registračním klíčem. Tento klíč je pak nutné vložit do druhé sekce formuláře a potvrdit jej.

### 3.7.2 Login page

Druhou ještě jednodušší stránkou je přihlašovací stránka. Zde uživatel vyplní své přihlašovací údaje a po jejich potvrzení je přesměrován do aplikace.

### 3.7.3 Basic info page

Tato stránka je nejdůležitější stránkou aplikace pro její správné fungování. Uživatel na této stránce ve formuláři podává veškeré informace o testovaném produktu. Tyto informace jsou po jejich potvrzení využity pro mnoho akcí prováděných na pozadí, tedy na back-endu. Pokud uživatel nevyplní tyto údaje, nedojde k napojení na API interních i externích systémů, uložení dat do databáze a tím pádem zůstane mnoho funkcí v aplikaci zcela nevyužitých. Po zadání správných údajů je uživatel přesměrován na první podstránku uvedenou v navigačním menu a tato podstránka (basic info page) je zablokována. Uživateli není umožněno se k této stránce vracet a veškeré případné změny v nastavení je třeba provádět přes modální okno po kliku na tlačítko „Change Settings“ v hlavičce aplikace.




**Obr. 15 Nastavení celého testu**
Zdroj: Vlastní zpracování

### 3.7.4 Automated tests

Jedná se o první krok celého testovacího flow, který je i v rámci navigačního menu oddělený od ostatních kroků, jelikož z důvodu optimalizace celé flow poběží automatické testy paralelně s dalšími kroky. Uživateli je v rámci sekce „automated tests“ zobrazen podrobný návod na získání dat z GitHubu a především následné nastavení a spuštění testů.

Popis všech testů je uveden níže v této práci. Veškeré testy jsou psány v Javascript frameworku zvaném Cypress. Uživatel si vygenerované testy pak může jednoduše v Cypress UI spustit, zatímco bude dál postupovat v testovaní dle zadané flow.

### 3.7.5 System listing

V této sekci by mělo dojít ke zjištění, zda dané produkty a veškeré jejich varianty jsou zalistované ve všech požadovaných (interních i externích) systémech. Celá tato sekce se dělí na 4 další podčásti odpovídající názvům jednotlivých systémů.

Všechny čtyři podsekce mají z větší části formu checklistu s popisem věcí, které je třeba otestovat. Pro účely diplomové práce nebylo umožněno využít API těchto systémů k zobrazení požadovaných dat přímo v aplikaci. Výjimkou tohoto tvrzení je napojení se na API pricing portalu a zobrazení ikon produktů pro rychlejší test. Zároveň je možné se z aplikace prokliknout přímo na detail produktu ve všech systémech.

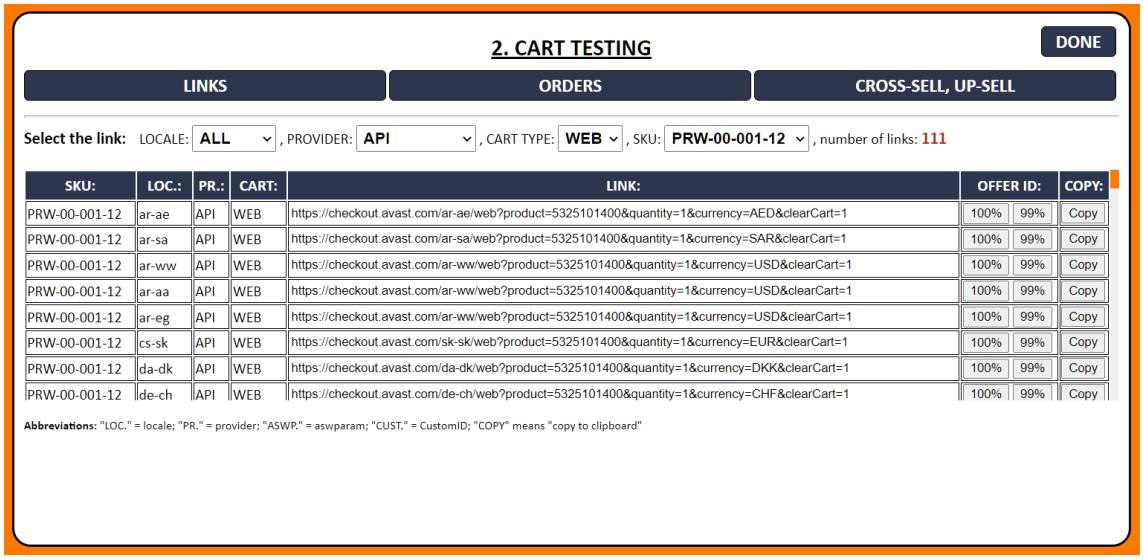
### 3.7.6 Cart testing

Stejně jako předchozí sekce je členěna na 4 části podle názvů systémů, tak tato sekce je též dále rozsegmentována, tentokrát dokonce na 5 podčástí.

- Links
- Orders
- Cross-sell, Up-sell

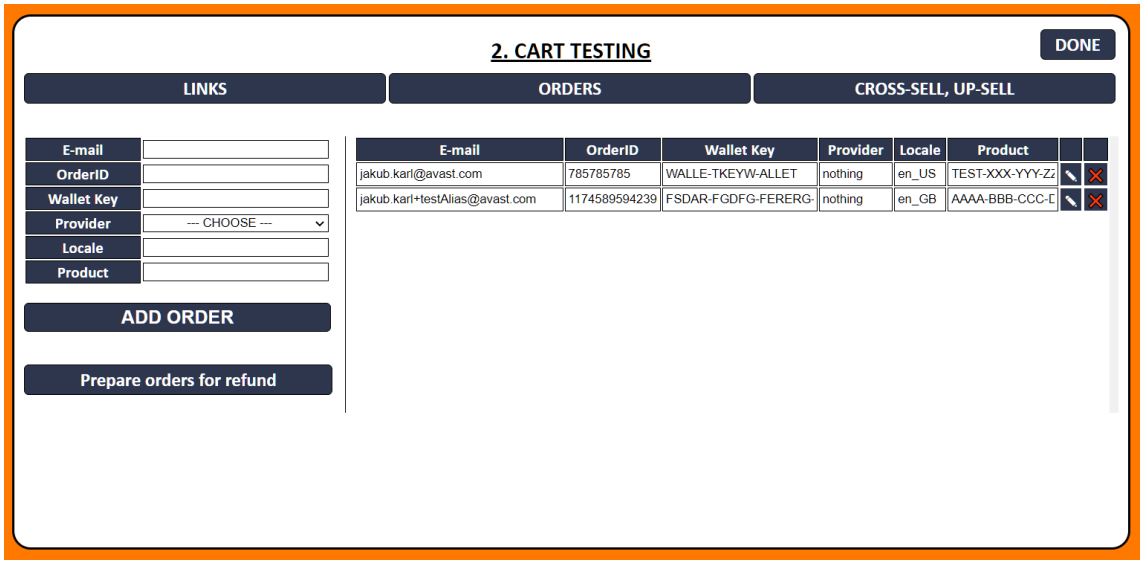
Jeden z nejdůležitějších testů celé workflow je otestování průchodu košíkem (e-shopem), všech elementů v košíku a i dokončení nákupu. Veškeré testovací úkony jsou vypsané v sekci „Cart Checklist“.

Pro snadnější získání linků do košíku lze využít sekci. V této sekci jsou zobrazeny všechny linky získané z API pricing portalu dle základního nastavení v sekci „basic info page“. Tyto linky je možné dále filtrovat (podle regionu, košíkového poskytovatele, produktové varianty, platformy apod.) díky automaticky vytvářeným selektorům.



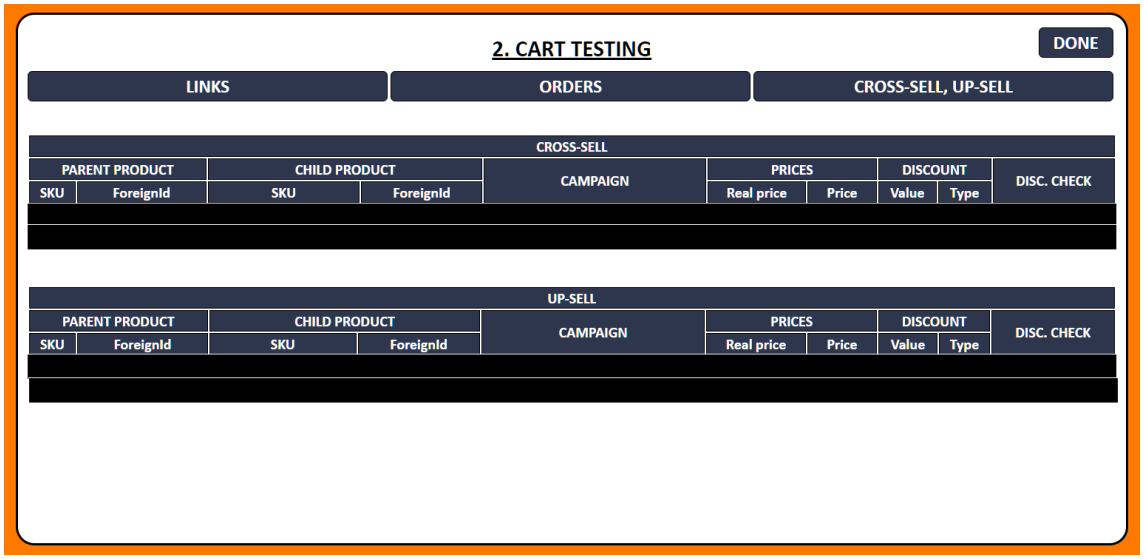

**Obr. 16 Stránka Cart testing, sekce Links**
Zdroj: Vlastní zpracování

Jak již bylo zmíněno, součástí testů je i dokončení nákupů. Objednávky je tedy dobré si zaznamenávat, jelikož později je třeba otestovat i různé parametry vytvořených objednávek a zároveň objednávky zrefundovat. K těmto účelům slouží podsekce „Orders“. Je zde možné objednávky přidávat, aktualizovat a odstraňovat. Veškeré změny jsou propsány do databáze.




**Obr. 17 Stránka Cart testing, sekce Orders**
Zdroj: Vlastní zpracování

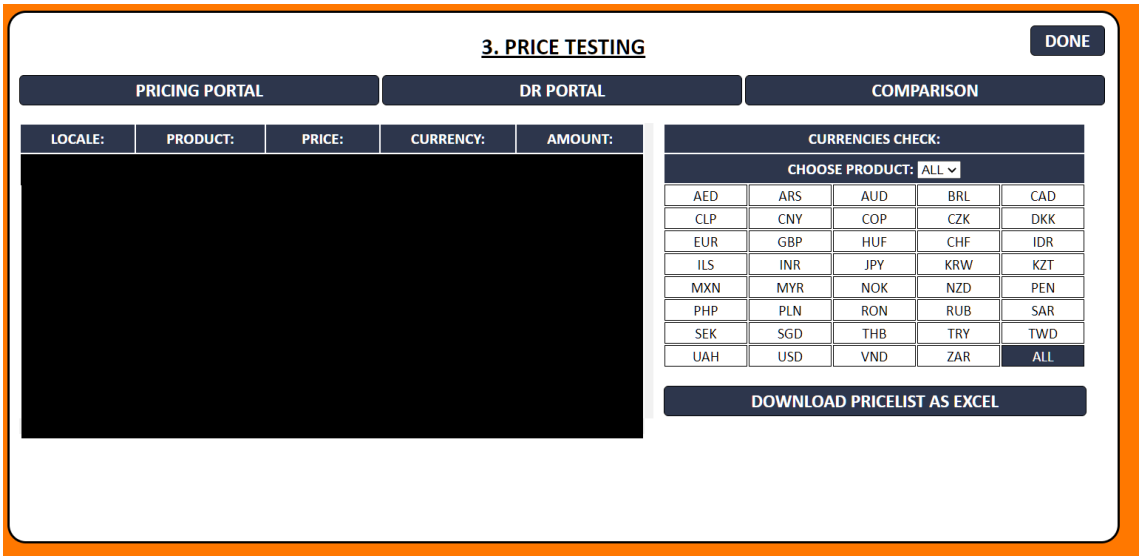
Poslední dvě sekce „Cross-sell“ a „Up-sell“ jsou téměř shodné. Nově vytvořený e-shop nazývaný API košík má implementované zobrazení modálních oken skrz který jsou zákazníkům nabízeny další produkty se slevou. V případě modelního okna spojeného s up-sellem je zákazníkovy nabídnuta „vyšší“ verze produktu (např. produkt využitelný pro více operačních systémů), který již v košíku má a původní produkt je odstraněn z košíku. V případě modálu spojeného s „cross-sell“ nabídkou je nový produkt přiložen do košíku k původnímu produktu a zákazník si může zakoupit oba. Obě tyto sekce se tedy zaměřují na testovaní těchto modálů. Opět je k tomu využito zobrazení dat z API pricing portalu.




**Obr. 18 Stránka Cart testing, sekce Cross-sell, up-sell, skrytá data**
Zdroj: Vlastní zpracování

### 3.7.7 Price testing

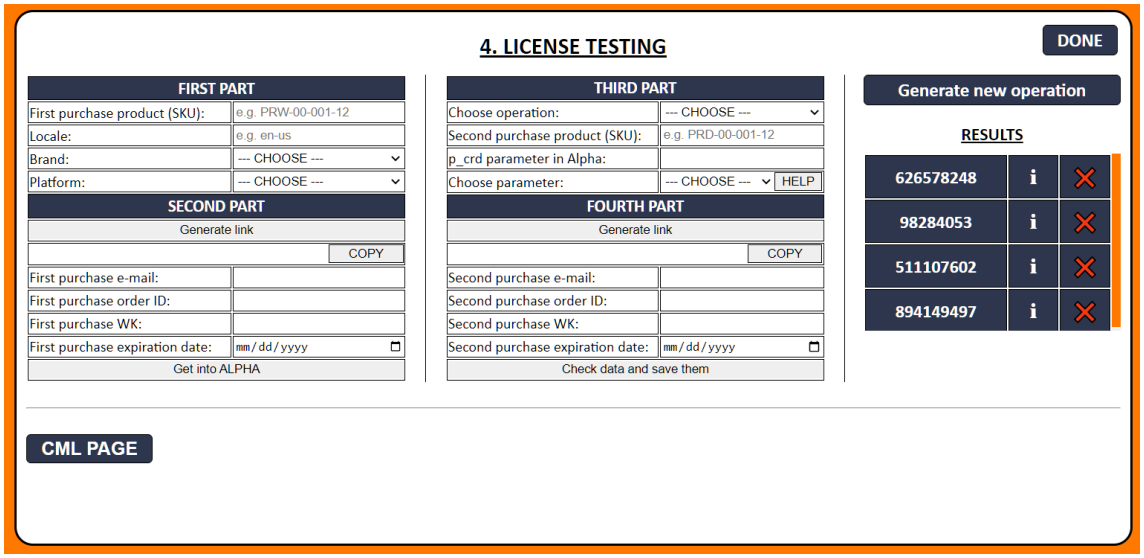
Ceny produktů jsou zadávány do dvou systémů. Do interního systému Pricing Portal a do externího systému poskytovatele košíku Digital River. Ceny v obou systémech musí být shodné a k ověření této hypotézy slouží právě tato sekce. Forma, jakou je sestaven tento test, je přes API obou systémů. Data z API jsou uložena do databáze, zobrazena na stránce a porovnávána. Ceny jsou roztřízeny dle poskytovatelů košíku, regionů, měny a dalších atributů a dle těchto parametrů mohou být filtrovány.




**Obr. 19 Stránka Price testing se skrytými daty**
Zdroj: Vlastní zpracování

### 3.7.8 License testing

U mnoha produktů je třeba testovat licenční operace. Tedy, jak se licence chová v případě její expirace či např. při „povýšení“ licence stávající na licenci „vyššího produktu“ (t.j. např. multiplatformní produkt či tzv. bundle, totiž produkt, jehož aktivací získá uživatel placenou verzi více aplikací). Pro tyto účely byl vytvořen formulář, který by měl usnadnit tyto testy. Uživatel zadá data potřebná k testu a generování URL adresy do e-shopu, zakódování parametrů pro správný chod testu, proklik do systémů a další operace zvládne již aplikace sama. Aplikace díky zadaným parametrům uživatele určí, zda daný typ licenční operace proběhl úspěšně či nikoliv. Samotný nákup však musí být zprocesován uživatelem, respektive za pomoci cypress testů. Všechny testy licenčních operací jsou ukládány automaticky do databáze, mohou být zobrazeny v modálním oknu, upraveny či smazány.




**Obr. 20 Stránka license testing**
Zdroj: Vlastní zpracování

### 3.7.9 E-mails

Tato stránka určená k seskupení všech informací spojených s testováním e-mailů je pravděpodobně nejméně propracovanou stránkou celé aplikace. Stránka obsahuje pouze checklisty, odkazy na externí nástroje využitelné pro testování notifikací s návody, jak tyto nástroje využít. Nejsou zde uvedeny žádné nové nástroje pro snadnější testování.

### 3.7.10 Orders testing

Sekce pro testování objednávek je spíše pomocná. Celá stránka je rozdělena na 2 části. V levé části je uveden checklist se všemi možnými testy napříč systémy. V pravé části se automaticky propisují potřebná data o objednávkách uložených v sekci „Cart testing“. Dále jsou zde přidána tlačítka pro snadnější proklik na stránky s detaily objednávek v rámci různých systémů.


**Obr. 21 Stránka Orders testing**
Zdroj: Vlastní zpracování

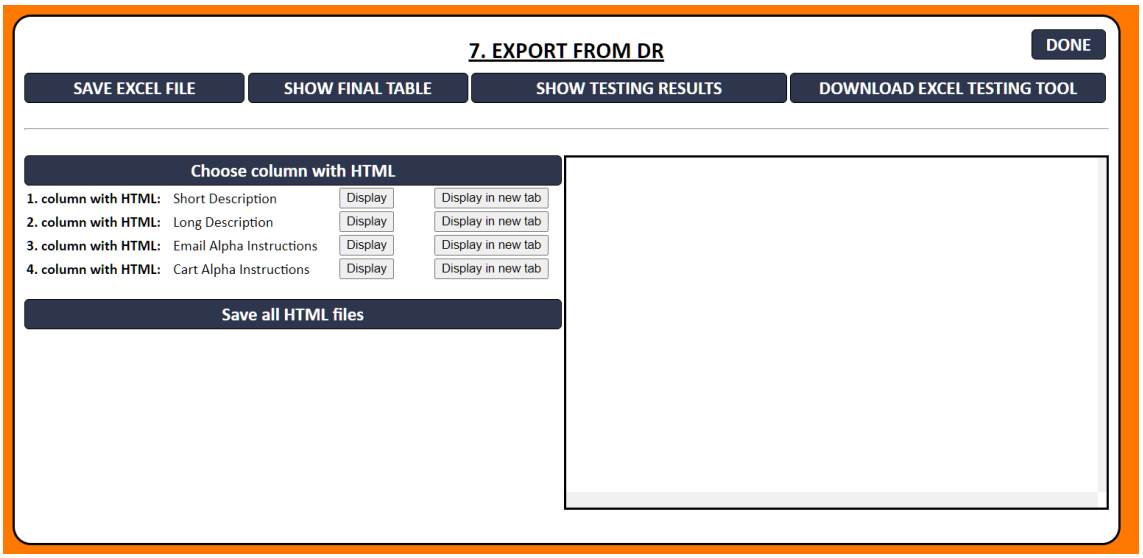
### 3.7.11 Export from DR

Důležitou součástí celé aplikace je zpracování exportu z již několikrát zmíněného externího systému poskytovatele košíku firmy Digital River. V Digital River portalu je nutné zalistovávat všechny produkty s poměrně širokým nastavením. Všechny testované atributy je možné vyexportovat do .xlsx souboru a tento soubor je pak testován. Na této podstránce by měl uživatel získat potřebné výstupy a nástroje pro správné otestování tohoto exportu. Exportované soubory mají jednotnou formu – obsahují vždy velké množství dílčích listů, které jsou též formátovány stejně (řádek s různými parametry produktu a mnoho dalších řádků podle počtu variant produktu s hodnotami těchto parametrů).

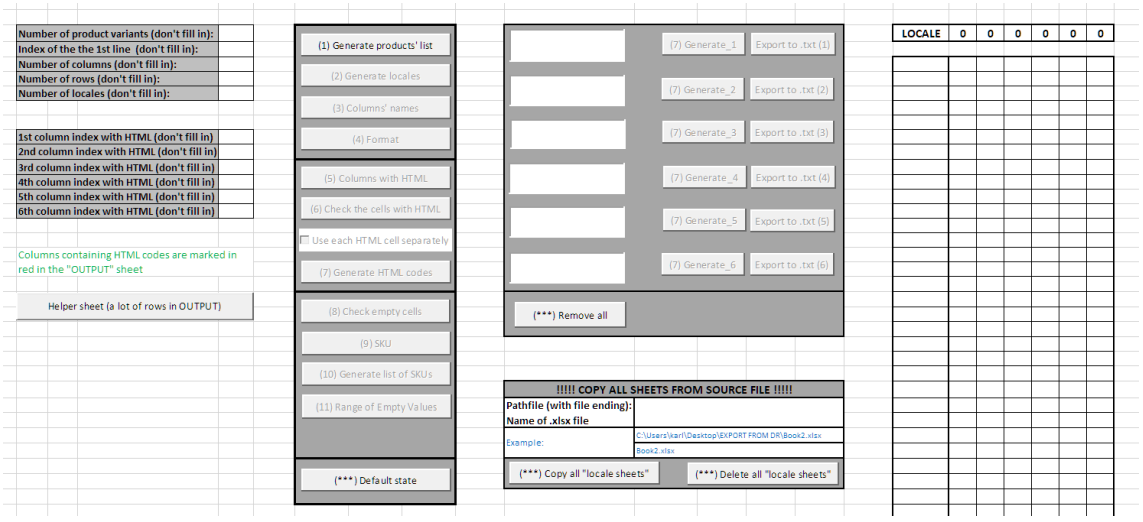
Všechna data z listů jsou v této aplikaci sloučena do jedné HTML tabulky, kterou je možné uložit v .xlsx formátu. Mnoho hodnot parametrů obsahuje HTML kódy. Tyto atributy jsou aplikací rozpoznány, „přeloženy“ z HTML a následně zobrazeny uživateli pro lepší a rychlejší testování. Tyto soubory je též možné stáhnout.

Všechny již zmíněné operace jsou nyní testovány za pomoci nástroje psaném v jazyce Visual Basic přímo v excelu. Protože byl tento nástroj vytvořen mnou, je

též přidán v této aplikaci, pokud by chtěl uživatel využít tuto starší formu, ačkoliv nebyl napsán pro účely diplomové práce.




**Obr. 22 Stránka Export from DR**
Zdroj: Vlastní zpracování




**Obr. 23 Excel nástroj pro zpracování exportu z DR**
Zdroj: Vlastní zpracování

## 3.8 Instalace a lokální spuštění cypress testů

Cypress testy, které byly napsané pro účely využití v rámci automatizace procesu testování nového produktu, jsou dostupné pouze na GitHubu. V aplikaci je kromě popisů testů uveden i podrobný návod, jak cypress lokálně nainstalovat, následně testy z GitHubu naklonovat do vlastního zařízení (uzpůsobeno operačnímu systému Windows) a poté i lokálně spustit. Aby bylo možné testy nastavit a spustit, je zapotřebí též vhodné IDE (zkratka pro integrated development environment, tedy software určený pro tvorbu aplikací kombinující obyčejné vývojářské nástroje a grafické uživatelské rozhraní. Uživatel může využít víceméně jakékoliv IDE, kde lze spustit Javascript a Node.js, avšak níže uvedené návody obsahují pokyny pro stažení Visual Studio Code. Číslované seznamy uvedené níže jsou psány jako návod pro uživatele aplikace. Jedná se v podstatě o instrukce přeložené z anglického jazyka (uvedené v aplikaci) do jazyka českého (pro účely diplomové práce).

Prvním krokem pro lokální (tj. na vlastním počítači) spuštění cypress testů je instalace Node.js. Node.js je velmi úzce spjato s balíčkovacím ekosystémem „npm“. Bez Node.js, respektive bez npm není možné nainstalovat cypress balíček. Uživatelé by měli dodržovat následující postup.

1. Na oficiálních webových stránkách https://nodejs.org/en/download/ v sekci „downloads“ stáhněte instalační soubor dle svého operačního systému.
2. Spusťte instalaci pomocí spuštění staženého instalačního souboru.
3. Postupujte dle instrukcí.

Druhým krokem je instalace IDE, v tomto případě již zmíněný a velmi oblíbený software Visual Studio Code.

1. Na oficiálních stránkách Visual Studio Code https://code.visualstudio.com/ stáhněte instalační soubor dle svého operačního systému.
2. Spusťte instalaci pomocí spuštění staženého instalačního souboru.
3. Postupujte dle instrukcí.

Následuje stažení software zvaného GitBash. Jedná se software pro komunikaci mezi GitHubem a lokální složkou s kódem.

1. Stáhněte si instalační soubor (pro operační systém Windows) z těchto webových stránek: https://gitforwindows.org/
2. Spusťte instalaci pomocí spuštění staženého instalačního souboru.
3. Postupujte dle instrukcí. Je možné potvrdit všechna výchozí nastavení.

Aby měl uživatel možnost využít testy z GitHubu, je potřeba naklonovat repozitář. To je možné provést například tímto způsobem.

1. Vytvořte složku kdekoliv v PC, kam chcete repozitář naklonovat
2. Otevřete nainstalovaný GitBash software
3. Za pomoci příkazu „cd“ je potřeba v GitBash připravit cestu do vytvořené složky, kam bude naklonován repozitář
4. Připojte se k VPN. Pro přístup do GitHubu je nutné být na interní síti firmy.
5. Otevřete si odkaz do repozitáře se cypress testy v GitHubu
6. V kmenové složce repozitáře klikněte na zelené tlačítko „Code“ a v záložce HTTPS zkopírujte link.
7. V GitBash aplikaci proveďte následující následující příkaz:
    - git clone ‘zkopírovanýLink‘
8. Zkontrolujte, zda soubory z repozitáře jsou již naklonovány ve složce
9. Otevřete si stažené IDE, tedy Visual Studio Code
10. V horní liště proveďte následující úkony: File -> Open Folder -> vyberte složku „cypress-tests“ ve Vámi vytvořené složce -> potvrdit
11. V horní liště proveďte následující úkony: Terminal -> New Terminal -> ve spodní části se objeví terminál
12. Nyní je třeba cypress nainstalovat do tohoto produktu. Zadejte do terminálu následující příkaz:
    - npm install cypress
13. Nyní je možné začít cypress používat. Cypress je možné spustit více způsoby.

- Jedním z nich je přes grafické uživatelské rozhraní. V takovém případě zadejte do terminálu příkaz např.:
    - npx cypress open
- Pokud máte raději výpisy do konzole a máte zájem o spuštění všech testů v rámci projektu, pak zadejte následující příkaz:
    - npx cypress run
- Pokud chcete spustit test s výpisy do konzole, avšak chcete spustit pouze jeden test, pak poslouží následující příkaz (textový řetezec „cestaKVybranémuTestu“ zaměňte za reálnou cestu k souboru, který chcete spustit):
    - npx cypress run –spec ‘./cestaKVybranémuTestu‘
- Testy je též možné spustit za pomoci předdefinovaných příkazů ve složce „commands“. V takovém případě stačí se za pomoci příkazu „cd“ dostat ke složce, kde se daný soubor s příponou .cmd nachází a následně do terminálu pouze uvést název tohoto souboru. Příkazy se spustí v pořadí v jakém jsou v souboru uvedeny.

## 3.9 Automatické testy

Všechny automatické testy využívané především pro testování košíku napříč poskytovateli, regiony, produkty apod. jsou psané v cypressu. Testy je možné si po jejich nastavení v aplikaci vygenerovat, stáhnout a spustit.

Celá tato sekce spojená s automatickými testy je mírně odlišná od všech ostatních sekcí v aplikaci. Automatické testy jsou totiž napsány tak, aby netestovaly pouze nový produkt, který je testován ve zbytku aplikace, ale bylo možné je spustit pro víceméně libovolnou konfiguraci produktů, regionů apod.

### 3.9.1 Testování cen

Aplikace obsahuje 2 testy spojené s testováním cen. Rozdíl mezi oběma testy je v typu košíků, které testuje. Jeden test je napsán pro tzv. API košík, tedy

košík, jejímž poskytovatelem je Digital River. Druhým typem košíku je košík od poskytovatele zvaného Softline. Struktura obou košíků je velmi odlišná, a právě z tohoto důvodu musí být i testy rozděleny do dvou.

První price test, tedy test API košíku je založen na porovnání cen z interního systému Pricing Portalu, externí systémů Digital River portalu a cen přímo v košíku. Data jsou uložena v JSON souboru. Cypress díky existanci API obou zmínených systémů dokáže zjistit ceny v těchto systémech a následně spustí dle nastavené konfigurace košíkovou část testu. Cypress projde všechny linky a zjištěné ceny uloží do již vytvořeného JSON souboru ke zbylým datům. Celý JSON je možné překonvertovat do excelovské podoby pro větší uživatelskou přehlednost.

Druhý zmíněný test, tedy test cen pro Softline košík probíhá víceméně totožně kromě absence získávání dat z Digital River portalu.

### 3.9.2 Testování modálních oken (cross-sell, up-sell)

Jedná se o další ze systémových testů. Velmi důležitou součástí API košíku jsou modální okna s výhodnými nabídkami zlevněných produktů. Při průchodu košíkem je po vyplnění základních údajů zákazníka nutných pro dokončení nákupu zobrazeno modální okno. V rámci tohoto okna je zákazníkovi nábídnut další produkt logicky související s produktem, který již v košíku je. Tento produkt je většinou značně zlevněný, je s ním spojena kampaň, díky níž je slevu možné uplatnit. Produkt je možné jednoduše přidat či odmítnout. Veškerá funkcionalita spojená s touto problematikou je testována tímto testem. Test otestuje, zda je modální okno zobrazeno, jaký je v něm nabízen produkt, je otestována cena produktu, sleva produktu, zda je možné produkt přidat. Jsou testovány i změny v URL adrese, propsání správné kampaně, navýšení ceny kompletního nákupu a spoustu dalších věcí. Výstupem testu jsou pak screenshoty modálního okna a JSON soubor s výsledky, které je možné jednoduše přeformátovat do excelu pro lepší čitelnost.

### 3.9.3 Screenshoter

Tento test má za cíl otestovat celou flow v košíku a z každého kroku udělat screenshot. Test je tedy určen pro testování především designových změn košíku a pro zjištění, jak je produkt v košíku zobrazen. Test má poměrně velké množství parametrů, díky nimž je možné nastavit, jaké kroky chce uživatel v košíku testovat.

### 3.9.4 Testování platebních metod

Avast prodává své produkty po celém světě. Z tohoto důvodu je tomu i košík uzpůsobený. V každém regionu jsou využívány jiné platební metody, ať už se jedná o online či offline způsoby plateb. Tento cypress test je určený k tomu, aby zvládl udělat výčet platebních metod ke každému regionu, screenshoty jejich flow a u metod, u kterých to je možné, dokončil i nákup.

### 3.9.5 Testování 1-clicku

Další z testů ověřuje správný chod funkcionality košíku, které se říká 1-click. Jak již název vypovídá, jedná se o průchod košíkem za pomoci pouze jednoho kliknutí myši. Pokud uživatel již v minulosti provedl nákup a byl mu správně vytvořen uživatelský účet, kterým se přihlásí do jakékoliv Avast či AVG aplikace, je mu tato funkcionalita povolena a další nákupy jsou značně zjednodušeny. Zákazník má možnost vybrat si ze všech dříve použitých platebních metod, případně přidat metodu novou, avšak při této akci je 1-click flow zrušeno a nákup probíhá standardním způsobem.

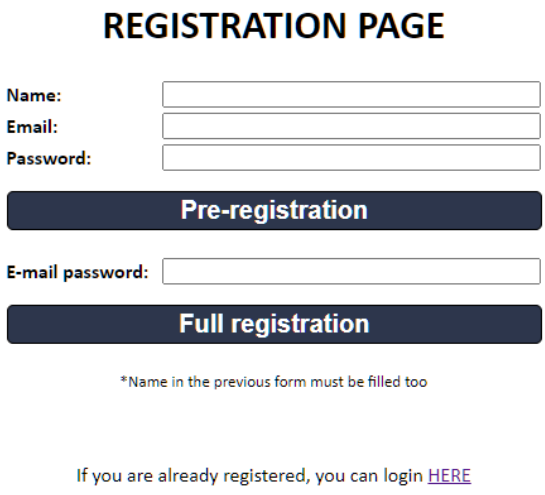
## 3.10 Zabezpečení aplikace

### 3.10.1 Autorizace

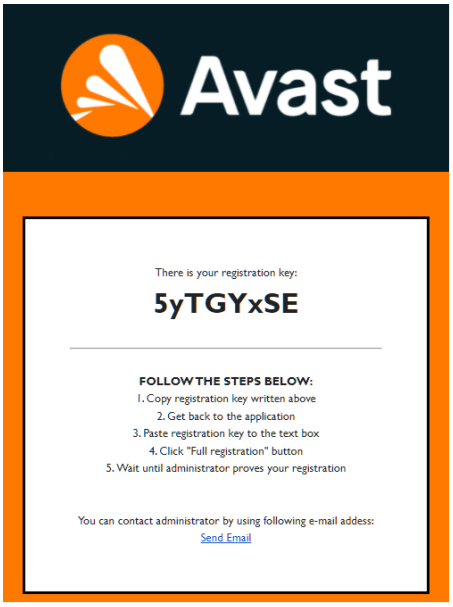
Jedním ze základních zabezpečovacích způsobů je autorizace. V aplikaci je vyvinut jednoduchý registrační a přihlašovací systém. Pouze přihlášený uživatel má přístup k datům v aplikaci. Pokud by se uživatel snažil např. přes URL adresu

dostat na dílčí stránky aplikace, nebude mu to umožněno a bude přesměrován zpět na přihlašovací stránku.

Proces registrace má tři fáze. První z nich je vyplnění základního přihlašovacího formuláře obsahujícího údaje jako přihlašovací jméno, e-mailová adresa a heslo. Hesla jsou vždy zašifrována pomocí hashování, aby nemohlo dojít k jejich zneužití. Podmínkou pro registraci je použití domény „@avast.com“ v e-mailové adrese. Jiné domény nejsou podporovány. Všechny tyto údaje jsou uloženy v databázi společně s dalšími automaticky vygenerovanými údaji jako je datum a čas registrace či za pomoci jednoduchého algoritmu vygenerovaný registrační kód, který je využíván v druhé fázi registrace. Ta je založena na odeslání registračního kódu na uvedenou e-mailovou adresu uživatele. Pro odeslání e-mailu je využita knihovna nodemailer a nastavení v Google Cloud Platform. Jakmile uživatel obdrží registrační kód, je nanavigován k jeho vložení zpět do aplikace a potvrzení. Pokud se zadaný kód shoduje s vygenerovaným kódem, je splněna další podmínka pro registraci a tento údaj je uložen k dalším datům o uživateli do databáze. Třetí fází registrace je potvrzení administrátorem. Každý uživatel musí mít účet schválený.




**Obr. 24 Registrační formulář**
Zdroj: Vlastní zpracování




**Obr. 25 Registrační notifikace**
Zdroj: Vlastní zpracování

### 3.10.2 VPN zabezpečení

Aplikace je z velké části založena na zobrazování dat z jiných interních systémů za pomoci API. Příkladem je zobrazení košíkových linků či cen ze systému zvaný Pricing Portal. Je zde využito interní API, které je přístupné pouze na interní síti, tedy přes VPN. Bez VPN nedojde k připojení k internímu API, a tím pádem je i funkčnost aplikace značně omezena.

### 3.10.3 HTTPS protokol

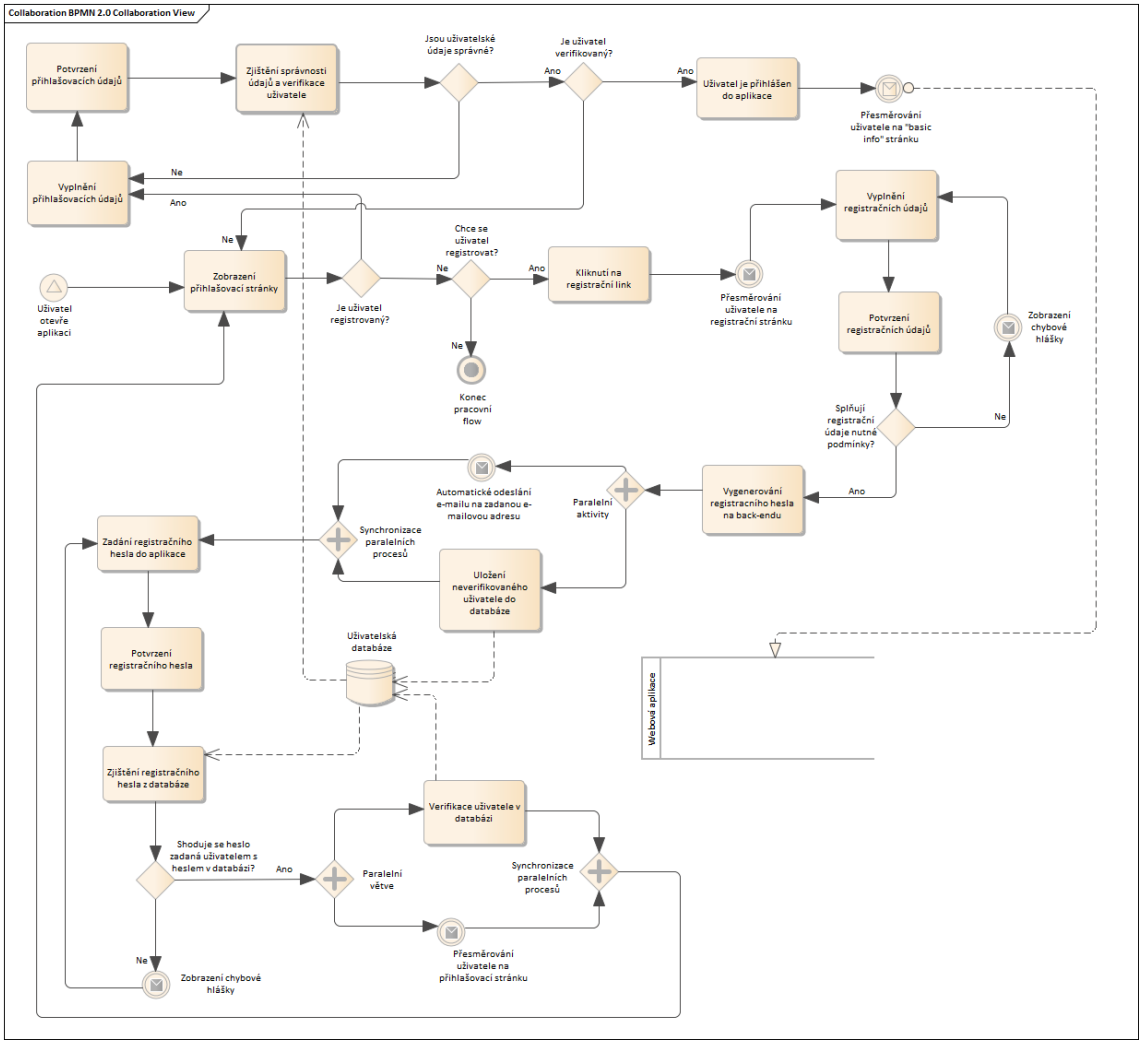
Jedním ze základních způsobů zabezpečení je využití HTTPS protokolu umožňující zabezpečenou komunikaci v počítačové síti. HTTPS protokol využívá protokol HTTP společně s protokolem SSL nebo TLS. Aplikace je naprogramovaná, aby využívala tento protokol, nicméně pro účely diplomové práce nebylo nutné přikládat validní certifikát. Validní certifikát bude dodán až při její integraci ve firmě.

## 3.11 Objektové modelování

Ačkoliv objektové modelování, konkrétně jazyky UML a BPMN, nabízí mnoho diagramů a schémat pro modelování funkčnosti aplikace nebo procesu, v diplomové práci byl využit pouze jeden diagram, a to konkrétné collaboration diagram zachycující registraci a přihlášení uživatele do aplikace. V rámci vývoje aplikace, respektive při jejím návrhu, se jednalo o jeden z nejtěžších úkolů, který bylo třeba naprogramovat, respektive vystihnout jednotlivé kroky. Z tohoto důvodu se jedná o jediný diagram.

Co se týče UML, většina z nejznámějších diagramů, byla zavrhnuta hned v počátcích. Takovým příkladem může být analytický a návrhový model tříd a sekvenční diagram. Důvod je jednoduchý – aplikace neobsahuje žádné třídy, které by bylo možné modelovat. Od začátku se počítalo s tím, že aplikace bude založena na funkcionálním programování. Tentýž důvod je v případě sekvenčního diagramu, který znázorňuje komunikaci tříd v čase. Více zamyšlení přinesl Use Case diagram. Ten byl výsledně též zavrhnut z důvodu, že případy užití (use cases) aktérů byly od

začátku známé a jasně specifikované, tudíž práce na tomto diagramu by byla více méně zbytečná.




**Obr. 26 BPMN schéma přihlášení a registrace uživatele**
Zdroj: Vlastní zpracování

## 3.12 Zdrojový kód

### 3.12.1 Adresářová struktura

Každá aplikace je v podstatě „pouze“ seskupení mnoha souborů a složek. Struktura složek a souborů nemá nikdy pevně dané náležitosti, avšak určité názvy

se u aplikací vyskytují opakovaně. Adresářová struktura testovací webové aplikace vyvíjené v této práci je následující:

- controllers
- css
- https
- images
- node_modules
- scripts
- views
- .env
- .gitignore
- app.js
- package.json
- passport-config.js

Hlavními složkami celé aplikace jsou složky controllers a scripts. Tyto složky obsahují pouze Javascript soubory s koncovkou .js. Rozdíl mezi soubory v obou složkách je ve vrstvě aplikace, kterou dané soubory spravují. Složka „controllers“ obsahuje soubory pro správu serverové části, tedy back-endu. Naopak složka „scripts“ obsahuje soubory s klasickým Javascriptem běžícím na klientovi v prohlížeči.

Ve složce „views“ nalezneme .ejs soubory, tedy soubory s nakódovanou strukurou celé aplikace. Z důvodu zvoleného view engine, tedy EJS, není možné vytvářet klasické HTML soubory, avšak pouze EJS soubory. HTML soubory nelze v aplikaci zobrazit. Jsou využívané pouze v případě, kdy je aplikací zpracováván soubor nahraný uživatelem do aplikace. Takovým příkladem může být zpracování exportu (v excelovské podobě) z externího systému DR, který obsahuje velké množství HTML kódu. Takový soubor ve své surové podobě se těžce kontroluje uživateli, jež má s HTML bohaté zkušenosti, natož pak uživateli, který tyto zkušenosti nemá. Aplikace z tohoto důvodu kód zpracuje a uloží jako HTML stránku. Tu si pak uživatel může snadno zobrazit či stáhnout a zkontrolovat v přeložené podobě. Jak již bylo uvedeno, HTML soubory nelze zobrazit v rámci

aplikace, jejíž view engine je nastavený na EJS soubory, naopak však EJS soubory nelze snadno zobrazit jako samostané webové stránky v prohlížeči. Z tohoto důvodu je využíván formát HTML.

Složka „css“ obsahuje veškeré soubory s kaskádovými styly, jež dotvářejí vzhled stránce. Složka images pak obsahuje png a jpg soubory, tedy obrázky zobrazované v aplikaci. Pouze dva soubory jsou umístěny ve složce https. Jedná se o certifikát a privátní klíč pro podporu https protokolu. Pro účely diplomové práce však nebyli vytvářeny validní soubory z důvodu vysoké ceny. Celé prostředí je však připraveno, aby případné validní soubory a tím i https protokol, podporovalo.

V sekci „node_modules“ nalezneme veškeré balíčky ve výchozím nastavení dané softwarovým systémem Node.js či později doinstalované. Na základě těchto balíčků je možné na serveru provádět různé operace, které Node.js poskytuje.

Zbylé body ve výše uvedeném seznamu nemají již podobu složek, ale souborů. První zmíněný, tedy .ejs, obsahuje nastavení prostředí, např. data v aplikaci využívaná, která by neměla být obsažena přímo v kódu z důvodu bezpečnosti. Soubor .gitignore zabraňuje při commitech nahrání určitých souborů do systému Git. Těmito soubory mohou být například Node.js balíčky či již zmíněný .env soubor.

Soubor app.js je pravděpodobně nejdůležitějším souborem celé aplikace. Jedná se totiž o spustitelný soubor aplikace. App.js soubor pak na sebe navazuje všechny další soubory pro správný běh aplikace na serveru. Je v něm uvedeno mnoho základních nastavení jako je view engine, statické soubory (css, images, scripts) atd.

Posledním souborem v celém výčtu je package.json. Jedná se, jak je na první pohled znatelné díky příponě souboru, o JSON soubor, který musí být umístěn v kořenovém adresáři projektu. Jsou v něm základní údaje o projektu včetně závislostí na nainstalovaných balíčcích.

V dalších níže uvedených sekcích jsou představeny důležité součásti ve zdrojovém kódu aplikace. Jedná se o „představitele“ důležitých struktur. Aplikace je však složena z tisíců řádků kódu, který nebude zde v práci uveden.

### 3.12.2 Obecné nastavení aplikace

V rámci aplikace je velmi důležitým souborem „package.json“, kde je uvedena základní konfigurace aplikace a závislosti na node modulech.

```
{
  "name": "testingapp",
  "version": "1.0.0",
  "description": "Application for 'product testing' process.",
  "main": "app.js",
  ▷ Debug
  "scripts": {
    "devStart": "nodemon app.js"
  },
  "keywords": ["qa","avast", "node.js", "javascript", "express", "pnp", "cart"],
  "author": "Jakub Karl",
  "license": "ISC",
  "dependencies": {
    "alert": "^5.0.10",
    "bcrypt": "^5.0.1",
    "body-parser": "^1.19.0",
    "convert-excel-to-json": "^1.7.0",
    "copy-dir": "^1.3.0",
    "crypto": "^1.0.1",
    "dotenv": "^10.0.0",
    "ejs": "^3.1.6",
    "express": "^4.17.1",
    "express-fileupload": "^1.2.1",
    "express-flash": "0.0.2",
    "express-session": "^1.17.2",
    "fs-extra": "^10.0.0",
    "https": "^1.0.0",
    "jsonwebtoken": "^8.5.1",
    "method-override": "^3.0.0",
    "mongoose": "^5.13.3",
    "nodemailer": "^6.7.2",
    "passport": "^0.5.0",
    "passport-local": "^1.0.0",
    "xmlhttprequest": "^1.8.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.13"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/JakubKarl/testingAppDiploma.git"
  }
}
```

**Obr. 27 package.json soubor**
Zdroj: Vlastní zpracování

### 3.12.3 GET requesty

Ačkoliv back-endová část aplikace obsahuje velké množství kódu, tak velkou a důležitou část tvoří tzv. get requesty. Ty slouží hlavně pro „renderování“ neboli propsání dat z back-endu, především z databáze, do „view“ aplikace uživateli. Get requesty v této aplikaci vypadají velmi podobně, liší se především daty, která jsou poskytována uživateli v rámci jednotlivých stránek aplikace. Příkladem může být níže uvedený kód pro propsání dat z kolekce „basicInformationmodels“ v databázi a dat ukládaných uživatelem do poznámek. Rozdílem mezi zápisem obou dvou případů je ten, že v jednom případě je v souboru schéma dokumentu ukládaného do databáze, kdežto v druhém případě nikoli.

```
app.get('/register', checkNotAuthenticated, (req, res) => {
    MongoClient.connect(url, function(err, db) {
        if (err) throw err;
        let dbo = db.db("ZkouskovyTicket");

        dbo.collection("basicinformationmodels").find({}).toArray(function(err, result) {
            if (err) throw err;

            NotesModel.find({}, function(err2, data) {
                if (err2) throw err;
                res.render('register', {
                    allBasic: {
                        pageName: req.url,
                        basicinformationmodels: result,
                        notesmodels: data,
                        name: "Not logged in yet",
                        action: req.query.action,
                    }
                });
            });
            db.close();
        });
    });
});
```

**Obr. 28 Get request pro registrační stránku**
Zdroj: Vlastní zpracování

### 3.12.4 POST requesty

Druhým nejčastějším „typem kódu“ jsou post requesty. Ty na rozdíl od get requestů mají opačnou funkci – data zapsaná uživatelem do aplikace jsou přes post requesty ukládány do databáze.

```
app.post('/saveNotes', urlencodedParser, function(req, res) {
    //delete all the previous documents in collections
    NotesModel.find({}).remove(function(err, data) {
        if (err) throw err;
    });

    //get data from the view and add it to mongodb
    NotesModel(req.body).save(function(err, data){
        if (err) throw err;
        res.json(data);
        console.log('Notes in database updated');
    });
});
```

**Obr. 29 Post request pro uložení poznámek**
Zdroj: Vlastní zpracování

### 3.12.5 AJAX

Back-end a front-end aplikace jsou, jak již název vypovídá, dva odlišné konce aplikace. Aby spolu oba konce mohly „komunikovat“, je v aplikaci využit tzv. Ajax. Ten může sloužit k mnoha věcem, avšak v této aplikaci je využíván především pro to, aby získal data od uživatele na front-endu, předal je post requestu na back-endu, který s daty může dále pracovat.

Následující kód na obrázku níže je příkladem Ajaxu. Ve chvíli, kdy je načtena stránka v aplikaci a uživatel klikne na element s id „workspace_database“, jsou konkrétní data vypsaná uživatelem zpracována pomoci funkce „createBasicInfoJson()“, jež, jak je z názvu patrné vrací objekt ve formě JSONu. Takovýto objekt je uložen do proměnné „allInfos“ a předán Ajaxu, který data předá POST requestu a díky asynchronnosti může po úspěšném vykonání požadavku vykonat další kód, v tomto případě pouze výpis dat do konzole.

```
$(document).ready(() => {
    $('#workspace_database').on('click', () => {
        const allInfos = createBasicInfoJson();

        $.ajax({
            type: 'POST',
            url: '/basic',
            data: allInfos,
            success: (data) => {
                console.log(data);
            }
        });

        return false;
    });
});
```

**Obr. 30 Ajax request**
Zdroj: Vlastní zpracování

### 3.12.6 Front-end Javascript

Aplikace obsahuje opravdu velké množství front-end Javascript kódu, jež je využíván především pro práci s DOMem, tedy pro úpravu jednotlivých elementů po akci uživatele. Dále je využíván pro zpracování dat, která jsou ukládána do databáze, zobrazování dat uživateli a mnoho dalších činností.

Javascript je možné pojmout dvěma způsoby – objektově a funkcionálně. V případě objektového přístupu jsou vytvářeny třídy s atributy a metodami. Tento přístup však v aplikaci není využit a je používán v podstatě čistě funkcionální přístup. Jednotlivé funkcionality aplikace jsou zapisovány do funkcí, které jsou pak volány při akci uživatele.

Kód níže není typickou funkcí, jedná se přidání event listeneru při načtení stránky, jež získává parametry z URL adresy a na základě toho zobrazuje požadovanou sekci stránky.

```
window.addEventListener("load", () => {
    switch (pageParam) {
        case 'links': document.getElementById('links').click(); break;
        case 'orders': document.getElementById('ordersButton').click(); break;
        case 'crossSell': document.getElementById('crossSellUpSell').click(); break;
    }

    document.getElementById('links').addEventListener('click', () => {
        params.set('page', 'links');
        window.location.search = params.toString();
    });

    document.getElementById('ordersButton').addEventListener('click', () => {
        params.set('page', 'orders');
        window.location.search = params.toString();
    });

    document.getElementById('crossSellUpSell').addEventListener('click', () => {
        params.set('page', 'crossSell');
        window.location.search = params.toString();
    });
});
```

**Obr. 31 Přidání event listeneru**
Zdroj: Vlastní zpracování

### 3.12.7 jQuery

Ačkoliv jQuery je velmi využívanou knihovnou jazyka Javascript, tak v aplikaci nemá příliš velké zastoupení. jQuery je využíváno pouze při tvorbě Ajax requestu a pro formátování stránky a pro „slide“ elementů DOMu.

```
$(document).ready(function(){
    $("#flip_1").click(function(){
        $("#panel_1").slideToggle("slow");
        $("#panel_2").slideUp("slow");
        $("#panel_3").slideUp("slow");
        $("#panel_4").slideUp("slow");
    });
});
```

**Obr. 32 jQuery formátování stránky**
Zdroj: Vlastní zpracování

## 3.13 SWOT analýza procesu produktového testování a testovací aplikace

| | POZITIVNÍ | NEGATIVNÍ |
|---|---|---|
| VNITŘNÍ PŮVOD | **SILNÉ STRÁNKY** | **SLABÉ STRÁNKY** |
| | • kompletní pokrytí celé flow produktového testování<br>• velké pokrytí testovacího procesu automatickými cypress testy<br>• vysoká míra použití<br>• rychlost aplikace<br>• jednoduché doplnění dodatečných pomocných nástrojeů do aplikace | • nevalidní HTTPS certifikát<br>• neresponzivní design<br>• nemožnost automatizace celého procesu<br>• některé části kódu mohou být pravděpodobně napsány přehledněji (důvodem je první zkušenost vývojáře s full-stack vývojem webové aplikace) |
| VNĚJŠÍ PŮVOD | **PŘÍLEŽITOSTI** | **HROZBY** |
| | • rozšíření o funkcionality potřebné k testování business produktů<br>• napojení nových funkcionalit v případě poskytnutí nových přístupů a oprávnění<br>• implementace již existujících open-source nástrojů do procesu | • nepodporování MongoDB databáze firmou Avast<br>• zastarání kódu<br>• změna integrovaných API endpointů a struktury requestů |

**Tabulka 2 SWOT analýza testovací aplikace**
Zdroj: Vlastní zpracování

# 4 Shrnutí výsledků a závěr

Teoretická část by se dle rozebíraných okruhů dala rozdělit do 3 částí. První část je zaměřena na optimalizační metody a optimalizaci procesu jako takovou, druhá pak seznamuje s obecnými znalostmi spojenými s testováním a poslední sekce představuje technologie využité v praktické části pro vývoj aplikace.

Hlavním cílem celé práce bylo zoptimalizovat již existující testovací proces zvaný „produktové testování“ především prostřednictvím webové aplikace, která by celý proces měla zaštiťovat, zjednodušovat a poskytovat jejímu uživateli kvalitní pracovní prostředí, aby bylo možné vytvořit co nejlepší výstup a odhalit co nejvíce chyb a v neposlední řadě automatické cypress testy, které zvládnou s minimálním úsilím uživatele odhalit velké množství chyb a nedostatků.

Praktická část práce pak v zásadě kopíruje základní strukturu části teoretické, respektive využívá informace z teoretické části. Prvním krokem pro naplnění stanovených cílů diplomové práce byla potřeba analýza stávajícího stavu procesu, aby bylo následně možné tento proces zrychlit, zlepšit a celkově zoptimalizovat. Pro tuto činnost bylo využito teoretických poznatků, které byly aplikovány do praxe. Příkladem může být jednoduchá SWOT analýza současného stavu procesu. Z rozboru současného stavu procesu a SWOT analýzy vyplynulo několik zásadních bodů, které bylo potřeba zlepšit, a na které se především zaměřovaly další kroky v rámci praktické části. Těmito body jsou sekvenčnost činností a téměř nulová automatizace.

Předpokládanými výsledky po optimalizaci procesu je menší náročnost testování díky zapojení automatizace a nástrojů pro snadnější odbavení dílčích kroků v testovacích scénářích, paralelní odbavování testovacích případů a rozšíření testovacích scénářů, případně objemu testovaných dat. Dále je třeba určitě uvést zrychlení kroků celého procesu, což je však v daném případě diskutabilní záležitost. Proces bude určitě rychlejší, ale nemusí dojít k jeho celkovému zkrácení. Pokud by nedošlo k poslednímu zmíněnému bodu, tedy zvětšení objemu testovaných dat, měl by být proces jednoznačně kratší. Vzhledem k tomu, že rozšíření testů je jedním z cílů práce, nemusí pak délka celého procesu být menší. Všechny zmíněné body jsou však pouze předpoklady a spekulace.

Práce představuje ucelený návod, jak je možné ve firmě Avast implementovat automatické testování vytváření nových produktů v katalogu a tento proces tak zefektivnit. Vyhodnocení míry zlepšení efektivity procesu je již nad rámec této práce.

Hlavním „hmotným“ výstupem pak kromě vylepšeného procesu, je testovací aplikace. Aplikace splňuje všechna očekávání navržená před jejím samotným vývojem a pravděpodobně se tak stane důležitou součástí při budoucím testování. Aplikace kopíruje hlavní kroky testovacího procesu produktového testování a vnáší do nich prvky zjednodušující testovací proces. Ne všechny hlavní kroky bylo možné zautomatizovat z důvodu neposkytnutí všech informací a přístupů pro účely diplomové práce kvůli možnému úniku citlivých dat. Tyto kroky pak mají formu jednoduchého nástroje pro usnadnění testovacích scénářů či pouze checklistu s kroky daného testu.

Ačkoliv aplikace je hlavním výstupem praktické části, tak není však výstupem jediným. Pro účely efektivnějšího testování byly napsány automatické testy v testovacím frameworku zvaném cypress. Tyto testy vnášejí do celého procesu již zmíněnou možnost testování většího objemu dat (příklad může být snadnější protestování košíku v rámci všech regionů oproti původnímu testování několika málo namátkou vybraných regionů). Díky integraci cypress testů bylo možné využít další zmíněné optimalizace – paralelizace procesu. Automatický test stačí spustit a během něj je možné provádět další manuální testy.

K celkovému zhodnocení procesu a aplikace byla využita SWOT analýza. Z ní je zřetelné, že aplikace má spoustu důležitých silných stránek, zároveň několik slabých stránek, které však nejsou pro funkčnost aplikace zásadní nebo se dají poměrně snadno odstranit. Aplikace ve stavu, v jakém je nyní, pravděpodobně do budoucna nezůstane. Jak je ze SWOT analýzy patrné, je možné aplikaci dále rozšiřovat, upravovat a zlepšovat.

# 5 Seznam použité literatury


[1] Avast About us | About AVAST Software. [online]. Dostupné z: https://www.avast.com/about

[2] Avast Software – Wikipedie. [online]. Dostupné z: https://cs.wikipedia.org/wiki/Avast_Software

[3] JESTON, John a Johan NELIS. Business process management: practical guidelines to successful implementations. Amsterdam: Elsevier, 2006. ISBN 0-7506-6921-7.

[4] CARDA, Antonín a Renata KUNSTOVÁ. Workflow: nástroj manažera pro řízení podnikových procesů. 2. rozš. a aktualiz. vyd. Praha: Grada, 2003. Management v informační společnosti. ISBN 80-247-0666-0.

[5] Process Optimization Methods: Definition, Benefits and Types | Indeed.com. Job Search | Indeed [online]. Copyright © 2022 Indeed [cit. 20.04.2022]. Dostupné z: https://www.indeed.com/career-advice/career-development/process-optimization-methods

[6] Process Optimisation in 5 Easy Steps. SolveXia - Low-Code Automation, Data Management and Analytics [online]. Dostupné z: https://www.solvexia.com/blog/process-optimisation-in-5-easy-steps

[7] FORTUNY, Raúl Peralba a Eduardo FAYOS-SOLÀ. SWOT analysis. JAFARI, Jafar a Honggen XIAO, ed. Encyclopedia of Tourism [online]. Cham: Springer International Publishing, 2016, 2016-6-25, s. 921-922 [cit. 2022-04-24]. ISBN 978-3-319-01383-1. Dostupné z: doi:10.1007/978-3-319-01384-8_537

[8] 7 Reasons Why Software Testing is Important. Indium Software – Digital Engineering Expertise that Makes Technology Work [online]. Copyright © 2022 All Rights Reserved [cit. 20.04.2022]. Dostupné z: https://www.indiumsoftware.com/blog/why-software-testing/

[9] GRIESKAMP, Wolfgang a Carsten WEISE, ed. Formal Approaches to Software Testing [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006 [cit. 2022-04-24]. Lecture Notes in Computer Science. ISBN 978-3-540-34454-4. Dostupné z: doi:10.1007/11759744

[10] O'REGAN, Gerard. Software Testing. O'REGAN, Gerard. Concise Guide to Software Engineering [online]. Cham: Springer International Publishing, 2017, 2017-05-31, s. 105-121 [cit. 2022-04-24]. Undergraduate Topics in Computer Science. ISBN 978-3-319-57749-4. Dostupné z: doi:10.1007/978-3-319-57750-0_7

[11] Dokumentace v testování | Testování softwaru. Testování softwaru [online]. Dostupné z: http://testovanisoftwaru.cz/dokumentace-v-testovani/

[12] Test Management. Test Management [online]. Copyright ©2021 [cit. 20.04.2022]. Dostupné z: https://www.testmanagement.com/

[13] BECK, Kent. Programování řízené testy. Praha: Grada, 2004. Moderní programování. ISBN 80-247-0901-5.SO

[14] Černá vs. bílá skříňka. Testování software [online]. Dostupné z: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=23:erna-vs-bila-skika&catid=3:zaklady&Itemid=11

[15] ACETOZI, Jorge. Types of Automated Tests. ACETOZI, Jorge. Pro Java Clustering and Scalability [online]. Berkeley, CA: Apress, 2017, 2017-08-11, s. 119-120 [cit. 2022-04-24]. ISBN 978-1-4842-2984-2. Dostupné z: doi:10.1007/978-1-4842-2985-9_23

[16] Automatizované testování softwaru | Testování softwaru. Testování softwaru [online]. Dostupné z: http://testovanisoftwaru.cz/automatizovane-testovani/

[17] Úrovně provádění testů | Testování softwaru. Testování softwaru [online]. Dostupné z: http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/faze-testu/

[18] Typy testování software. Radek Kitner - konzultant, lektor testování softwaru [online]. Dostupné z: https://kitner.cz/testovani_softwaru/typy-testovani-software-trideni-testu/

[19] What Is Sanity Testing? [with Examples] - PractiTest. PractiTest - QA Test Management Tool that works for You [online]. Copyright © 2022 H.S PractiTest. All rights reserved [cit. 24.04.2022]. Dostupné z: https://www.practitest.com/qa-learningcenter/best-practices/what-is-sanity-testing/

[20] CASTRO, Elizabeth a Bruce HYSLOP. HTML5 a CSS3: názorný průvodce tvorbou WWW stránek. 2. vydání. Přeložil Ondřej BAŠE, přeložil Kristýna BAŠE. Brno: Computer Press, 2022. ISBN 978-80-251-5045-0.

[21] EJS -- Embedded JavaScript templates. EJS -- Embedded JavaScript templates [online]. Copyright © 2112 [cit. 19.04.2022]. Dostupné z: https://ejs.co/

[22] Nestjs vs Express.js. Nestjs Tutorials [online]. 2020, Květen 2020 [cit. 19.04.2022]. Dostupné z: http://www.nestjstutorials.com/nestjs-vs-express-js/

[23] OLSSON, Mikael. JQuery. OLSSON, Mikael. JavaScript Quick Syntax Reference [online]. Berkeley, CA: Apress, 2015, 2015, s. 61-70 [cit. 2022-04-24]. ISBN 978-1-4302-6493-4. Dostupné z: doi:10.1007/978-1-4302-6494-1_15

[24] Mozilla Developer Network. MDN [online]. 2021 [cit. 19.04.2022]. Dostupné z: https://developer.mozilla.org/enUS/docs/Web/API/XMLHttpRequest/onreadystatechange

[25] FLANAGAN, David. JavaScript: kompletní průvodce. 2. aktualiz. vyd. Praha: Computer Press, 2002. Všechny cesty k informacím. ISBN 80-7226-626-8.

[26] ŠKULTÉTY, Rastislav. JavaSript: programujeme internetové aplikace. 2. aktualiz. vyd. Brno: Computer Press, 2004. ISBN 80-251-0144-4.

[27] ASLESON, Ryan a Nathaniel T. SCHUTTA. Ajax: vytváříme vysoce interaktivní webové aplikace. Brno: Computer Press, 2006. ISBN 80-251-1285-3

[28] What is MongoDB? A definition from WhatIs.com. Purchase Intent Data for Enterprise Tech Sales and Marketing - TechTarget [online]. Dostupné z: https://www.techtarget.com/searchdatamanagement/definition/MongoDB

[29] ŽÁRA, Ondřej. JavaScript: programátorské techniky a webové technologie. Brno: Computer Press, 2015. ISBN 978-80-251-4573-9.

[30] NGUYEN, Don. Node.js Okamžitě. Přeložil Ondřej BAŠE. Brno: Computer Press, 2016. ISBN 978-80-251-4820-4.

[31] Cypress Automation Step by Step: Cypress Architecture, Install Cypress - Lambda Geeks. Home - Lambda Geeks [online]. Copyright © 2022, LambdaGeeks.com [cit. 24.04.2022]. Dostupné z: https://lambdageeks.com/cypress-automation-cypress-architecture-install-cypress/

[32] JavaScript End to End Testing Framework | cypress.io testing tools. JavaScript End to End Testing Framework | cypress.io testing tools [online]. Copyright © [cit. 24.04.2022]. Dostupné z: https://www.cypress.io/

[33] Cypress Testing: How to Get Started | Perfecto. Web & Mobile App Testing | Continuous Testing | Perfecto [online]. Copyright © 2022 Perforce Software, Inc. [cit. 24.04.2022]. Dostupné z: https://www.perfecto.io/blog/cypress-testing

[34] What is UML | Unified Modeling Language. Welcome To UML Web Site! [online]. Copyright © 2022 [cit. 24.04.2022]. Dostupné z: https://www.uml.org/what-is-uml.htm

[35] KOSSAK, Felix, Christa ILLIBAUER, Verena GEIST, et al. A Rigorous Semantics for BPMN 2.0 Process Diagrams [online]. Cham: Springer International Publishing, 2014 [cit. 2022-04-24]. ISBN 978-3-319-09930-9. Dostupné z: doi:10.1007/978-3-319-09931-6

[36] BOEHM, Barry W. Software Engineering Economics. BROY, Manfred a Ernst DENERT, ed. Software Pioneers [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, 2002-10-21, s. 641-686 [cit. 2022-04-25]. ISBN 978-3-642-63970-8. Dostupné z: doi:10.1007/978-3-642-59412-0_38

# 6 Seznam obrázků

# 7 Seznam tabulek

# 8 Přílohy

Příloha č. 1

*Oskenované zadání práce*

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2020/2021

Studijní program: Informační management
Forma studia: Prezenční
Specializace/kombinace: Informační management (im2-p)

# Podklad pro zadání DIPLOMOVÉ práce studenta

| | |
|---|---|
| Jméno a příjmení: | **Bc. Jakub Karl** |
| Osobní číslo: | **I1900303** |
| Adresa: | Kostnická 328, Kolín – Kolín IV, 28002 Kolín 2, Česká republika |
| Téma práce: | Tvorba testovací aplikace, automatizace a optimalizace testovacího procesu |
| Téma práce anglicky: | Creating a test application, automation and optimization of the test process |
| Vedoucí práce: | doc. Ing. Filip Malý, Ph.D.<br>Katedra informatiky a kvantitativních metod |

Zásady pro vypracování:

Student vypracuje teoretickou a praktickou část diplomové práce dle zadané osnovy:

1. Úvod
2. Cíl práce
3. Teoretická část
4. Praktická část
5. Výsledky a závěr
6. Zdroje

Seznam doporučené literatury:

ŽÁRA, Ondřej. JavaScript: programátorské techniky a webové technologie. Brno: Computer Press, 2015. ISBN 978-80-251-4573-9.
NGUYEN, Don. Node.js Okamžitě. Přeložil Ondřej BAŠE. Brno: Computer Press, 2016. ISBN 978-80-251-4820-4.
Node.js – Serverový JavaScript. itnetwork.cz – Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další. [online]. Copyright © 2021 itnetwork.cz. Veškerý obsah webu [cit. 27.09.2021]. Dostupné z: https://www.itnetwork.cz/javascript/nodejs

Podpis studenta: Karl

Datum: 26.4.2022

Podpis vedoucího práce:

Datum: 27.4.2022