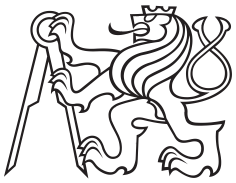
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Testování platformy pro tvorbu internetových obchodů

Jan Valeš

Školitel: doc. Ing. Ivan Jelínek, CSc.
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Valeš** Jméno: **Jan** Osobní číslo: **499049**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Testování platformy pro tvorbu internetových obchodů

Název bakalářské práce anglicky:

Testing the platform for creating online stores

Pokyny pro vypracování:

Cílem bakalářské práce je implementovat automatizované testy, které ověří správnou funkčnost kritických částí platformy používané k tvorbě internetových obchodů webovou agenturou Blueghost. Základním požadavkem na testy je snížení nákladů na výrobu obchodů skrze automatizované testování za těchto podmínek:
Testy budou spustitelné hromadně pomocí jediného příkazu.
Všechny testy musí být vyhodnoceny jako úspěšně, nebo v případě chyby, prověřeno, zda se jedná o chybu platformy.
Výstupem testů bude dostatečné množství snímků obrazovky, aby bylo možné chybu snadno identifikovat a duplikovat.
Testy budou snadno upravitelné, aby při vývoji nového obchodu nebylo nutné upravovat všechny testy, nýbrž nejmenší možné množství kódu testů.
Proveďte rešerši existujících technologií pro testování platformy, uvažte použití frameworku Symfony.
Proveďte testovací analýzu společně a plánem a strukturou testů k implementaci
Implementujte automatizované testy. Vytvořte min. 52 testů průchodu aplikací a volání API endpointů (konkrétně min. 17 testů pro manipulaci s košíkem (API), min. 10 na úpravu košíku při průchodu aplikací, min. 18 testů pro objednávkový formulář a 7 testů pro registrační formulář)
Vytvořte manuál pro spouštění a pro implementaci nových testů
Navrhněte způsob ověření korektnosti testování platformy a výsledek zhodnoťte

Seznam doporučené literatury:

[1] Certifikovaný tester základní úrovně, International Software Testing Qualifications Board, Verze 2018 v3.1
[2] Testing (Symfony Docs) [online]. Symfony™, 2021. [cit. 2022/09/27]. Dostupné z: https://symfony.com/doc/current/testing.html
[3] Bureš, M. et al. Efektivní testování softwaru: Klíčové Otázky pro Efektivitu Testovacího Procesu. Grada, 2016. ISBN 978-80-247-5594-6.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Ivan Jelínek, CSc. kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

doc. Ing. Ivan Jelínek, CSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Poděkování

Děkuji panu doc. Ing. Ivanu Jelínkovi, CSc za vedení projektu. Dále děkuji panu Ing. Michalovi Zímovi za technické konzultace a dohled.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 24. května 2023 .....
Jan Valeš

Abstrakt

Bakalářská práce se zabývá automatickým testováním platformy pro tvorbu internetových obchodů na míru webové agentury BlueGhost.cz, s.r.o. Platforma je napsaná v PHP frameworku Symfony. Cílem práce bylo automatizovat testování jejích kritických částí. Byla provedena analýza komponent platformy. Na základě analýzy byly vybrány kritické části, pro které byl vypracován plán testů technikou testování černé skříňky. Testy byly implementovány pomocí knihovny Panther.

Klíčová slova: Automatizované testování, PHP, Symfony, Selenium, Internetový obchod

Školitel: doc. Ing. Ivan Jelínek, CSc.
Katedra počítačů,
Fakulta elektrotechnická,
České vysoké učení technické v Praze,
12000 Praha 2

Abstract

This Bachelor's thesis focuses on the automatic testing of a custom e-commerce platform developed by the web agency BlueGhost.cz, s.r.o. Platform is built using PHP framework Symfony. The aim of the thesis was to automate the testing of critical parts of the platform. An analysis of the platform components was conducted, and based on this analysis, critical parts were identified and a test plan was developed using black box testing technique. The tests were implemented using the Panther library.

Keywords: Automated testing, PHP, Symfony, Selenium, Eshop

Title translation: Testing the platform for creating online stores

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Nechtěnou součástí vývoje softwaru je řešení chyb při jeho tvorbě. Některé chyby zůstávají neodhaleny a přenášejí se do dalších částí vývoje. Oprava chyb v pozdních částech vede zpravidla ke zvyšování nákladů. [3] Efektivní nástroj pro včasnou detekci chyb je automatické testovaní, které umožňuje spouštět celkovou kontrolu vyvíjeného softwaru při každé drobné úpravě.

Cílem práce je automatizovat testování kritických částí platformy pro tvorbu internetových obchodu na míru webové agentury BlueGhost.cz, s.r.o. Ta v současné době neobsahuje žádné testy. Absence automatických testů vede ke zvyšování nákladů na údržbu obchodů. Každá drobná změna kódu musí být důkladně otestována. Zároveň existuje riziko lidské chyby, kdy tester může chybu zcela přehlédnout. Implementací automatických testů by mělo dojít k šetření času testerů i snížení rizika nepovšimnutí lidské chyby.

1.1 Popis platformy

Platforma slouží jako základní stavební blok při tvorbě internetových obchodů na míru. Obsahuje jejich společné prvky a umožňuje jednoduchou implementaci nových funkcí dle požadavků klienta. Každý nový obchod je kopií této platformy.

1.1.1 Architektura platformy

Platforma se skládá ze dvou Symfony aplikací propojených skrze REST (Representational State Transfer) API. Jedna aplikace slouží jako frontend pro zobrazování dat uživateli. Veškerá data získává z druhé aplikace skrze API. Pouze druhá aplikace má přístup k databázi. Dokumentace API je přístupná v Apiary. [1]

Výhodou této architektury je možnost napojení několika frontendových aplikací na jednu backendovou. Díky tomu je možné spravovat více internetových obchodů v jediné backendové aplikaci, což umožňuje tvorbu několika podobných obchodů za nižší náklady.

[1] https://bgeshop3.docs.apiary.io

1.1.2 Použité technologie platformy

Platforma využívá standardní technologie pro vývoj webových PHP aplikací. Správných návrh plánu testů vyžaduje základní znalost technologií, které platforma využívá. Zde je jejich stručný popis:

HTML, CSS a JavaScript je sada nástrojů používaných při tvorbě webových stránek. HTML (Hypertext Markup Language) je značkovací jazyk. Zapisuje se skrze něj obsah webové stránky. Způsob zobrazení (vzhled) webové stránky definuje CSS (Cascading Style Sheets). Interaktivní prvky webu jsou řízeny skriptovacím jazykem JavaScript.

PHP je jednoduše naučitelný skriptovací jazyk určený k vytváření dynamických HTML stránek. Spouštění většinou probíhá na webovém serveru. Nejnovější verze platoformy využívá PHP7.4. [5]

MariaDB je relační databáze vycházející z MySQL. K práci z daty používá jazyk SQL (Structured Query Language).

Symfony je open-source framework určený pro tvorbu webových aplikací v PHP. Skládá se z řady komponent, které ulehčují jejich vývoj.

Doctrine ORM (object-relational mapper) poskytuje PHP persistentní ukládání objektů jako dat. Ulehčuje tak programátorům práci s jejich ukládáním. [6]

Webpack je Symfony komponenta ulehčující práci s CSS a JavaScriptem. Poskytuje nástroje pro pre-processing, kompilaci a minifikaci assetů webu. [7]

jQuery je JavaScriptová knihovna navržená pro zjednodušení práce s HTML strukturou a manipulací. Dalšími funkcemi jsou jednoduché animace a AJAX (Asynchronous JavaScript And XML) požadavky. [8]

Docker a Docker Compose jsou použity pro lokální vývoj obchodů. Docker je nástroj pro virtualizaci softwaru do tzv. "kontejnerů". V případě platformy jsou webový server a databáze samostatné kontejnery. Docker Compose slouží pro orchestraci kontejnerů. S jeho použitím je možné jednoduše spravovat aplikace využívající několik kontejnerů. Díky těmto nástrojům je možné vyvíjet více obchodů zároveň i v případě, že každý obchod využívá jiné verze technologií. Jednotlivé obchody se vzájemně neovlivňují a není potřeba řešit složitou správu několika verzí technologií v jednom systému. Platforma pracuje s následujícími kontejnery:

- Builder - slouží ke spouštění příkazů v obou aplikacích.
- Back a Front - v těchto kontejnerech běží backendová respektive frontendová aplikace na serveru Apache. Na rozdíl od Builderu je instalace minimalistická a neobsahuje Composer a další pomocné nástroje.
- MariaDB - kontejner ve kterém běží databáze.
- PHPMyAdmin - nástroj pro správu databáze.
- Traefik - reverzní proxy využívaná k přístupu do kontejneru přes URL, například *eshop3.local*.

Kapitola 2
Techniky testování

Bezchybnost softwaru lze ověřit úplným pokrytím všech možných stavů. Takové testování zpravidla není možné, nebo je nepřiměřeně náročné. Proto je potřeba stanovit přijatelnou úroveň chybovosti. Efektivního pokrytí s malou pracností lze dosáhnout volbou vhodné techniky testování. V této kapitole popisuji techniky testování dle jejich úrovně a přístupu.

2.1 Úrovně testování

Před volbou vhodných technologií pro testování je potřeba rozlišit úrovně testování. Každá úroveň je vhodná v jiné části vývoje softwaru. Jednotlivé úrovně vyžadují různé zdroje, proto se mohou lišit technologie použité pro jejich testování. Při správném testování dochází ke kombinování všech úrovní. [1]

2.1.1 Jednotkové testy

Jednotkové testování (Unit testing) se zaměřuje na testováních atomických částí kódu (jednotlivé funkce nebo metody), který je spouštěn samostatně a izolovaně od zbytku systému. Jednotkové testování je vhodné provádět hned během vývoje aplikace. Aby bylo možné izolovaně testovat jednotlivé metody, využívá se **mockování** (falešné objekty). V takovém případě se nejedná o skutečnou instanci objektu, nýbrž o objekt s předpřipravenými daty pro testování. Mockování umožňuje simulaci vnějších vlivů, jako je například databáze. [1]

2.1.2 Integrační testování

Cílem integračního testování je ověřit interakci mezi částmi aplikace, které předtím byly testovány samostatně. Integrační testy se zaměřují na ověření, zda různé komponenty správně spolupracují a integrují se navzájem. [1]

2.1.3 Systémové testování

V posledních částech vývoje aplikace lze přistoupit k systémovému testování. Během systémového testování dochází ke kompletní emulaci aplikace a testování probíhá, jako by ho prováděl uživatel. Takové testování může být pomalejší než jednotkové nebo integrační testování, jedná se však o nepostradatelný test aplikace jako celku. Výhodou takového testování je, že **není potřeba znalost kódu** (Testování černé skříňky). [1]

2.1.4 Akceptační testování

Akceptační testování se zaměřuje na fungování systému jako celku. Jeho cílem není odhalování chyb, nýbrž posouzení stavu projektu před dokončením. Nalezení velkého množství chyb v této fázi může být považováno za významné riziko. Akceptační testování není předmětem automatizovaných testů. [1]

2.2 Testování bíle skříňky

Testování bíle skříňky vychází ze znalosti architektury a způsobu zpracování uvnitř objektu. Cílí na pokrytí všech částí kódu, případně všech rozvětvení kódu. Mohou tak velice důsledně najít možné chyby v kódu. Vyžadují však výbornou znalost kódu testovaného softwaru. [1]

2.3 Testování černé skříňky

Testování černé skříňky se zaměřuje na chování výsledné aplikace z pohledu uživatele. Tester neřeší vnitřní zpracování aplikace. Není proto nutná detailní znalost testovaného kódu. Technika pracuje s pouze očekávanými vstupy a výstupy aplikace.[1] [2]

2.3.1 Rozdělení tříd ekvivalence

Rozdělení tříd ekvivalence je technika, při které se vstupní data rozdělí do tříd, pro které je očekáváno stejné chování. Správného pokrytí se dosahuje v případě, že je použita minimálně jedna hodnota z každé třídy ekvivalence. [1]

Plné pokrytí více vstupů by znamenalo otestovat všechny kombinace tříd ekvivalence jednotlivých vstupů. To vede k prudkému nárůstu počtu testů, které je potřeba uskutečnit. Snížení počtu testů lze docílit technikou **pairwise** (párového) testování. Cílem této techniky je otestovat všechny páry vstupních hodnot.

Kapitola 3

Analýza komponent platformy

V této kapitole se věnuji jednotlivým komponentám platformy. Pro každou komponentu určuji pravděpodobnost selhání a vážnost následků v případě výpadku. Pravděpodobnost selhání určuji na základě: *složitosti funkcí komponenty, velikosti kódu, historických problémů s danou komponentou, frekvenci úprav komponenty a vlastních zkušenostech.* Možné poškození určuji na základě konzultace s technickým ředitelem agentury BlueGhost a vlastních zkušenostech.

3.1 Produkty

Přidávání produktů probíhá v administraci. Důležité položky nastavované u produktu jsou: *Kategorie, Cena, DPH, Aktivní, Akce, Doprava zdarma, Priorita, Kusy skladem, Obrázky.* Za selhání se považuje, pokud se produkt zobrazí zákazníkům jinak, než dle nastavení v administraci. Takové selhání může mít velký vliv na reputaci obchodu. Vzhledem ke stejnému zdroji dat je pravděpodobnost selhání nízká.

Možné poškození: Vysoké Pravděpodobnost selhání: Nízká

3.2 Košík

Košík slouží pro vytváření objednávek. Ukládají se v něm produkty zvolené uživatelem. Historicky v něm docházelo k chybám, proto se pravděpodobnost selhání stanovuje na střední. Špatné vkládání produktů do košíku představuje zásadní překážku v fungování internetového obchodu.

Možné poškození: Vysoké Pravděpodobnost selhání: Střední

3.3 Objednávka

Objednávka je nejdůležitější součástí internetového obchodu. Zákazník nejdříve zvolí způsob dopravy a platby. Pro vybrané země je od určité částky doprava zdarma. Zákazník následně nastaví dodací údaje a případně doplní poznámku či fakturaci na firmu. Dále má možnost zvolit, zda se chce zaregistrovat. Kvůli velkému množství zpracovávaných polí je pravděpodobnost selhání vysoká.

Možné poškození: Vysoké Pravděpodobnost selhání: Vysoká

3.4 Zákaznický účet

Zákazníci si mohou založit účet v sekci registrace, nebo během vytváření objednávky. Účet slouží ke sledování stavu objednávek a nastavení osobních údajů, které se použijí při tvorbě nové objednávky. Tuto funkci nevyužívají všichni zákazníci, z toho důvodu je možné poškození střední. Pravděpodobnost selhání je střední, protože historicky měli někteří uživatelé problém s přihlášením.

Možné poškození: Střední Pravděpodobnost selhání: Střední

3.5 Vypnutí webu

Administrace umožňuje okamžité vypnutí všech stránek internetového obchodu. To je důležité zejména v průběhu nasazení aplikace. Dále může sloužit jako záchranná brzda v případě kritické chyby. Funkce je implementována v jádře celé platformy. Pravděpodobnost selhání je tedy nízká.

Možné poškození: Vysoké Pravděpodobnost selhání: Nízká

3.6 SEO

Důležitou součástí internetového obchodu je SEO (optimalizace pro vyhledávače). Tu je možné provádět na několika místech v administraci. Každý produkt, kategorie a stránka mají nastavitelné SEO tagy pro poskytnutí informací vyhledávačům.

SEO je implementovaná funkcionalita bez důvodu budoucí úpravy. Jelikož nastavení provádí administrátor, dá se předpokládat, že veškeré změny kontroluje a nedojde selhání. Pro komplexní SEO analýzu existují jiné nástroje, které by správný specialista měl využívat. Nedává proto smysl provádět automatické testy této komponenty.

3.7 GTM

Google Tag Manager je nastaven majitelem internetového obchodu v administraci. Jedná se o oblíbený nástroj využívaný k měření konverzí a návštěvnosti. Vložení do obchodu probíhá na základě GTAG ID nastaveného v administraci. Funkce nemá vliv na zákazníky, proto je možné poškození nízké. Vzhledem k tomu, že načtení GTM probíhá externě na základě nastavení v administraci, je i pravděpodobnost selhání nízká.

Možné poškození: Nízké Pravděpodobnost selhání: Nízká

3.8 Přepravní kombinace

Administrace umožňuje nastavení zemí, plateb a dopravních služeb. Pro každou kombinaci těchto prvků lze nastavit DPH, Cenu a částku od které je přeprava zdarma. Zároveň je možné některou z těchto kombinací vypnout a uživateli tak znemožnit vytvoření objednávky s některou z kombinací.

Uživatel způsob platby a dopravy zadává při tvorbě objednávky. Frontendová aplikace si platnost této kombinace ověřuje skrze API.

Možné poškození: Střední Pravděpodobnost selhání: Střední

3.9 Jazykové mutace

Platforma podporuje až pět jazykových mutací internetového obchodu. Překlady se ukládají do databáze. Pro používání se pomocí příkazu kompilují do formátu JSON (JavaScript Object Notation) během nasazení na server. Jelikož jsou veškeré překlady zadávány administrátorem obchodu, lze předpokládat že správné zobrazení zadaných textů kontroluje. Pravděpodobnost selhání je proto nízká.

Možné poškození: Střední Pravděpodobnost selhání: Nízká

3.10 Měnové mutace

Platforma podporuje měny CZK a EUR. Cena produktů je vždy zadávána v CZK. Kurz pro převod EUR na CZK se nastavuje v administraci. Výsledná cena všech objednávek je ukládána v CZK společně s převodním kurzem v moment vytvoření objednávky. Vzhledem k způsobu implementace, který předchází chybám způsobeným volbou špatné měny, je pravděpodobnost selhání nízká.

Možné poškození: Střední Pravděpodobnost selhání: Nízká

Kapitola 4
Rešerše technologií

V analýze komponent platformy jsem určil potřebné funkcionality technologií pro testování. V této kapitole se věnuji průzkumu technologií, které vyhovují analýze komponent. Během průzkumu beru ohled zejména na jednoduchost implementace do již existující aplikace.

Automatické testování webové aplikace lze provádět v cela jiném jazyce, než ve kterém je napsána samotná aplikace. Testy totiž přistupují pouze k výsledné aplikaci. V BlueGhostu je hlavní jazyk pro tvorbu aplikací **PHP**, který všichni programátoři důvěrně znají. Bylo proto rozhodnuto, že testy budou implementovány ve stejném jazyce, aby budoucí úpravy a implementace testů nebyly překážkou pro ostatní programátory.

4.1 PHPUnit

Jednou ze součástí Symfony je integrace PHPUnit. Jako součást Symfony umožňuje bez další konfigurace jednoduché spouštění testů a mockování Symfony komponent. To dělá z PHPUnit ideální technologii pro provádění Unit testů v rámci Symfony aplikace. [4]

4.2 Symfony KernelTestCase

Symfony jako nástroj pro integrační testování využívá KernelTestCase. Jedná se o TestCase využívaný pro jednotkové testování, který je rozšířený o Dependency injection[1]. Tím je možné testovat několik plnohodnotných komponent a interakce mezi nimi. [4]

4.3 Symfony WebTestCase

Symfony má pro systémové testování připravený WebTestCase. Ten umí procházet stránky dle URL, hledat v obsahu stránky, následovat přesměrování a další funkce webových prohlížečů. Jedná se však pouze o simulaci pomocí

[1] Návrhový vzor pro vkládání objektů, bez nutnosti znalosti jejich vnějších závislostí.

PHP komponent, nikoli skutečný prohlížeč. Kvůli tomu neumožňuje odhalení veškerých chyb, například v Javascriptu. [4]

4.4 Selenium

Selenium je framework pro automatizaci testů webových aplikací. **Selenium WebDriver** jednoduché ovládání prohlížeče, jako by ho prováděl uživatel. Poskytuje API pro ovládání webových prohlížečů, jako například Firefox, Opera, Google Chrome, nebo Safari.[10] **Selenium Grid** umožňuje paralelní spouštění testů na více uzlech (zařízeních). Centrálním bodem je Hub, který přeposílá žádosti do správných uzlů. Poskytuje přehledné rozhraní pro sledování průběhu testů a vytíženosti uzlů. [11]

4.5 Panther

Panther je knihovna určená přímo pro použití v Symfony. Implementuje napojení na Selenium WebDriver. Rozšiřuje tak Symfony WebTestCase o emulaci opravdového webového prohlížeče. Díky integraci se Symfony se Panther stává ideálním nástrojem pro systémové testy. [4]

Kapitola 5

Plán testů

V analýze komponent platformy byla určena pravděpodobnost selhání a možné poškození v případě selhání pro všechny komponenty platformy. V této kapitole je definován plán testů pro nejdůležitější komponenty platformy, které byly zvoleny na základě konzultace s technickým ředitelem agentury BlueGhost.

5.1 Košík

Obsluha košíku probíhá skrze tři jednoduché endpointy. Ve frontendové části aplikace jsou řízeny velkým množstvím javascriptu, ve kterém historicky docházelo k chybám. Při testování je proto třeba brát důraz na systémové testy obou aplikací.

5.1.1 Košík API testy

Endpoint pro úpravu produktů v košíku má tři metody: set, add a sub, přičemž pro odebrání produktu z košíku se používá metoda set s nulovým počtem kusů. Vrácená odpověď je ve formátu JSON. Ten obsahuje aktualizované informace o produktech a objednávce. Testován bude správný obsah a struktura vrácené odpovědi.

U každé metody, kde to je možné, budu testovat následující případy:

- Přidání nového produktu v košíku kladným číslem
- Přidání nového produktu v košíku záporným číslem
- Úprava množství existujícího produktu kladným číslem
- Úprava množství existujícího produktu záporným číslem
- Nastavení množství existujícího produktu na výslednou hodnotu < 0
- Nastavení množství neexistujícího produktu na výslednou hodnotu < 0
- Operace s nevalidním produktem

Metoda set

- testBasketSet_emptyBasket_itemAdded()
- testBasketSet_emptyBasket_negativeQuantityErrorReturned()
- testBasketSet_itemInBasket_itemQuantityIncreased()
- testBasketSet_itemInBasket_negativeQuantityErrorReturned()
- testBasketSet_invalidProduct_nothingChanged()

Metoda add

- testBasketAdd_emptyBasket_itemAdded()
- testBasketAdd_emptyBasket_negativeQuantityReturned()
- testBasketAdd_itemInBasket_itemQuantityIncreased()
- testBasketAdd_itemInBasket_negativeQuantityErrorReturned()
- testBasketAdd_invalidProduct_nothingChanged()

Metoda sub

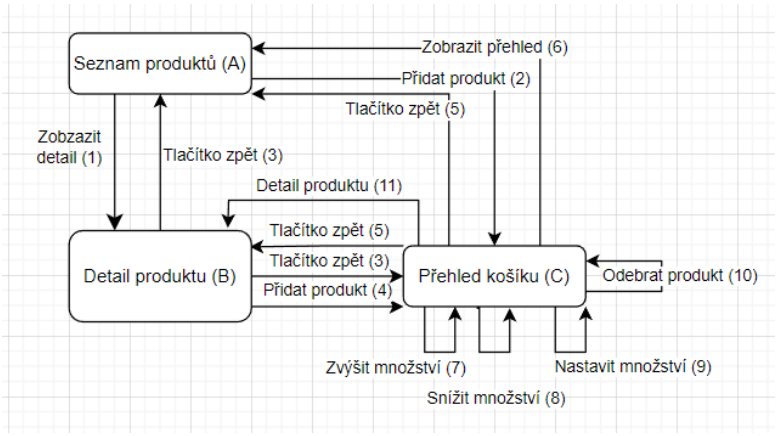
- testBasketSub_emptyBasket_nothingChanged()
- testBasketSub_emptyBasket_negativeQuantityErrorReturned()
- testBasketSub_itemInBasket_itemQuantityDecreased()
- testBasketSub_itemInBasket_negativeQuantityErrorReturned()
- testBasketSub_itemInBasket_itemRemoved()
- testBasketSub_itemNotInBasket_nothingChanged()
- testBasketSub_invalidProduct_nothingChanged()

5.1.2 Systémové testy košíku

Uživatel může produkty do košíku přidávat a ubírat ve třech částech aplikace. Průchod uživatele lze znázornit jednoduchým diagramem. Cílem je otestovat, že některá z operací nenaruší strukturu dat košíku a nezpůsobí tak chybu při budoucích krocích.

V diagramu 5.1 můžeme vidět stránky, ze kterých lze spravovat obsah košíku. Stránky jsou označeny písmeny A až C. Pro každou stránku jsou uvedeny operace označeny čísly 1 až 11. Nejdříve určím kombinace vstupních a výstupních akcí pro každý bod.

V bodě C tabulky 5.1 dochází ke vzniku velkého počtu kombinací. Pro jeho snížení použiji předpoklad, že druh operace s košíkem nemá vliv na průchod. Díky tomu mohu body 7 až 10 spojit do jednoho ve výsledné tabulce 5.2. Body 7 až 10 mohu testovat samostatně.



Obrázek 5.1: Diagram průchodu košíkem.

Větvící bod	Vstupní akce	Výstupní akce	Kombinace pro test
A	3, 5, 6	1, 2	3-1, 3-2, 5-1, 5-2, 6-1, 6-2
B	1, 5, 11	3, 4	1-5, 1-4, 5-3, 5-4, 11-3, 11-4
C	2, 3, 4, 7, 8, 9, 10	5, 6, 7, 8, 9, 10, 11	49 kombinací!

Tabulka 5.1: Tabulka kombinací možných akcí.

Větvící bod	Vstupní akce	Výstupní akce	Kombinace pro test
A	3, 5, 6	1, 2	3-1, 3-2, 5-1, 5-2, 6-1, 6-2
B	1, 5, 11	3, 4	1-3, 1-4, 5-3, 5-4, 11-3, 11-4
C	2, 3, 4, (7-10)	5, 6, (7-10), 11	2-5, 2-6, 2-(7-10), 3-5, 3-6, 3-(7-10), 4-5, 4-6, 4-(7-10), 4-11, (7-10)-5, (7-10)-6, (7-10)-(7-10)

Tabulka 5.2: Tabulka kombinací možných akcí se spojenými body.

Průchod košíkem nemá žádný koncový stav. Uživatel může přejít k vyplnění objednávky v libovolném bodě. Počáteční operace musí být přidání produktu, tedy stav 2 nebo dvojice 1-4. Délku testu lze volit libovolně tak, aby bylo docíleno nejvyšší efektivity programátora. Jako střed mezi příliš dlouhými testy a velkým množstvím testů jsem se rozhodl pro délku testu o pěti krocích.

Test	Sekvence akcí
1	A-2-5(A)-2
2	B-3(A)-1-4-6
3	A-2-(7-10)-5(A)
4	B-3(A)-2-6-1
5	B-4-5(B)-3(A)-2
6	A-2-(7-10)-(7-10)
7	B-4-5(B)-4-(7-10)
8	B-4-11-4-6
9	A-2-11-3(C)-(7-10)
10	B-4-11-3(C)-5(B)
11	A-2-5(A)-1-3(A)
12	A-2-(7-10)-6
13	B-4-5(B)-3(A)-6

Tabulka 5.3: Tabulka scénářů průchodu košíkem.

Pro testování operací 7 až 10 na detailu košíku vzniká 16 kombinací. Na začátku každého testu jsou do košíku vloženy dva produkty.

Test	Sekvence akcí
1	A-2-6-2-7-7-8-7-9-7-10-7
2	A-2-6-2-8-8-9-8-10-8
3	A-2-6-2-9-9-10-9
4	A-2-6-2-10-10

Tabulka 5.4: Tabulka scénářů úpravy obsahu košíku.

5.2 Proces objednávky

Každé pole vyplněné uživatelem je validováno skrze API. Není možné formulář odeslat, pokud některé z polí není vyplněno správně.

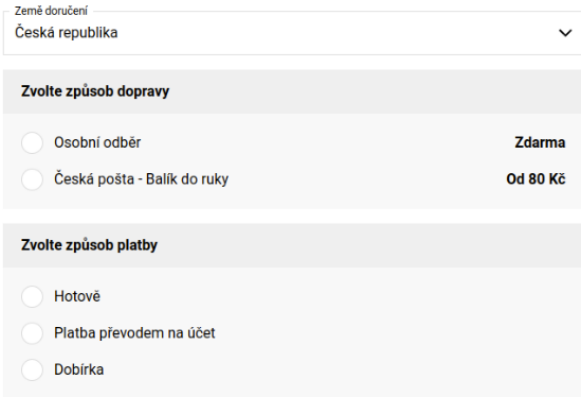
5.2.1 Validace jednotlivých polí objednávky

Správnou kontrolu vstupních dat formuláře lze provádět skrze API. Pro každé vstupní pole formuláře určím validní a nevalidní třídy ekvivalence, pro které budu provádět testy.

Doprava a platba

Způsob dopravy a platby si uživatel může zvolit na základě hodnot připravených v administraci obchodu. Základními možnostmi dopravy jsou *osobní odběr* a *Česká pošta*. Platba může být zvolena *hotově*, *převodem* nebo *na účet*. Některé z výsledných nedávají smysl a jsou proto **zakázány**. Například není

možné zvolit doručení Českou poštou a platbu v hotovosti. Při volbě osobní odběr se uživateli zakáže možnost zvolit platbu hotově.



Obrázek 5.2: Formulář doprava a platba.

Vstupní pole	Validní třída ekvivalence	Nevalidní třída ekvivalence
Země doručení	Existující hodnota selectu (1, 2)	Neexistující hodnota selectu (-1, 0, 10)
Způsob dopravy	Existující hodnota výběru (1, 2)	Neexistující hodnota výběru (-1, 0, 10)
Způsob platby	Existující hodnota výběru (1, 2, 3)	Neexistující hodnota výběru (-1, 0, 10)

Tabulka 5.5: Tabulka tříd ekvivalence formuláře dopravy a platby.

Osobní údaje

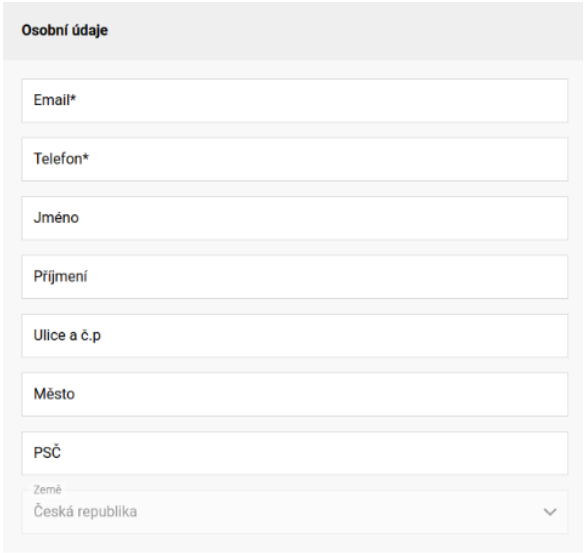
Platforma neprovádí pokročilou validaci pro pole *Telefon, Jméno, Příjmení, Ulice a č.p, Město, PSČ*. Validuje pouze přítomnost těchto polí. Nevalidní třídou ekvivalence je prázdná hodnota. Validní třída ekvivalence obsahuje všechny řetězce o libovolných znacích.

Nákup na firmu

Platforma neprovádí pokročilou validaci pro pole *Název, IČO, DIČ*. Validuje se pouze přítomnost těchto polí a nezáleží na jejich hodnotě. Nevalidní třídou ekvivalence je prázdná hodnota. Validní třída ekvivalence obsahuje všechny řetězce o libovolných znacích.

Obchodní podmínky

V sekci obchodní podmínky uživatel může zvolit pole, zda se chce zaregistrovat. Dále musí zaškrtnout souhlas s obchodními podmínkami a se zpracováním osobních údajů. Pole jsou **povinné** a bez souhlasu nelze objednávku odeslat.



Obrázek 5.3: Formulář osobní údaje.

Vstupní pole	Validní třída ekvivalence	Nevalidní třída ekvivalence
Email - délka	1 - 254 znaků	0 znaků, >= 255 znaků
Email - formát	x@y.y, kde x je 1-64 znaků a y je 1-63 znaků	x o délce 0 a 65 nebo více znaků, y o délce 0 a 64 nebo více znaků
Telefon, Jméno, Příjmení, Město, Ulice a č.p, PSČ	Neprázdná hodnota	Prázdná hodnota
Způsob platby	Existující hodnota výběru (1, 2, 3)	Neexistující hodnota výběru (-1, 0, 10)

Tabulka 5.6: Tabulka tříd ekvivalence formuláře osobní údaje.

5.2.2 Komplexní funkčnost

V předchozí části jsem pomocí testů ověřil, že každé vstupní pole obsahuje pouze validní data. Nyní je možné otestovat kompletní proces objednávky formou UI testů. V tabulce 5.9 níže se nacházejí hodnoty, pro které dává smysl testovat vytvoření objednávky. Testy provádí průchod celým procesem vytvoření nové objednávky.

Metodou pairwise jsem vygeneroval 58 scénářů, pomocí kterých dosáhnu požadovaného pokrytí. Tabulka scénářů je k nalezení na přiloženém CD pod názvem *"objednavka-pairwise.xlsx"*. Součástí každého testu bude kontrola správného uložení dat.

Vstupní pole	Validní třída ekvivalence	Nevalidní třída ekvivalence
Název, IČO, DIČ	Neprázdná hodnota	Prázdná hodnota

Tabulka 5.7: Tabulka tříd ekvivalence formuláře nakupuji na firmu.

Vstupní pole	Validní třída ekvivalence	Nevalidní třída ekvivalence
Registrace	Ano, ne	-
Souhlas s obchodními podmínkami	Ano	Ne
Souhlas se zpracováním osobních údajů	Ano, ne	-

Tabulka 5.8: Tabulka tříd ekvivalence formuláře obchodní podmínky.

Vstupní pole	Hodnoty pro testování	Poznámka
Země doručení	Česká republika, Slovensko	
Celková cena	< Částka pro dopravu zdarma, > Částka pro dopravu zdarma, Částka pro dopravu zdarma	Celková cena se vypočítá na základě košíku, není možné aby byla záporná
Způsob dopravy	Osobní, Česká pošta	
Způsob platby	Hotově, převodem, dobírka	
Doručení na jinou adresu	Ano, ne	
Nákup na firmu	Ano, ne	
Poznámka	Ano, ne	
Registrace	Ano, ne	
Měna	CZK, EUR	

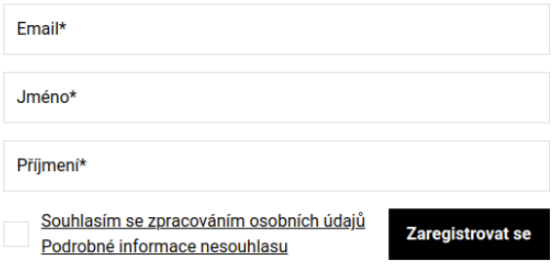
Tabulka 5.9: Tabulka tříd ekvivalence objednávkového formuláře.

5.3 Registrace

Formulář registrace obsahuje čtyři vstupní pole. Jediné validované pole ve formuláři je email. Jeho správnost je ověřována pomocí testu API.

Registrační formulář je možné pokrýt několika systémovými testy. V tabulce 5.10 jsou data, pro která dává smysl registraci testovat.

Úspěšné registrace lze dosáhnout pouze dvěma scénáři. Jméno a příjmení musejí být vyplněny. Email musí být ve správném formátu. Souhlas s obchodními podmínkami je volitelný. Cílem pozitivních testů je vyzkoušet, zda je možné se zaregistrovat se souhlasem i bez něj. K neúspěšnému pokusu o registraci dojde, pokud je email ve špatném formátu nebo není vyplněno jméno. Výsledné scénáře se nacházejí v tabulce 5.11.



Obrázek 5.4: Formulář registrace.

Vstupní pole	Hodnoty pro testování
Email	Validní, Nevalidní délka, Nevalidní formát, Duplicitní
Jméno	Prázdné, neprázdné
Příjmení	Prázdné, neprázdné
Souhlas	Ano, ne

Tabulka 5.10: Tabulka hodnot pro testování registračního formuláře.

Scénář	Email	Jméno	Příjmení	Souhlas	Úspěch
1	Validní	Neprázdné	Neprázdné	Ne	Ano
2	Validní	Neprázdné	Neprázdné	Ano	Ano
3	Nevalidní délka	Neprázdné	Neprázdné	Ne	Ne
4	Nevalidní formát	Neprázdné	Neprázdné	Ano	Ne
5	Duplicitní	Neprázdné	Neprázdné	Ano	Ne
6	Validní	Prázdné	Neprázdné	Ne	Ne
7	Validní	Neprázdné	Prazdné	Ano	Ne

Tabulka 5.11: Testovací scénáře registračního formuláře.

Kapitola 6
Implementace

V této kapitole popisuji průběh. První část obsahuje přípravu vývojového prostředí a potřebné úpravy aplikace ke spouštění testů. V druhé části jsou popsány použité technologie a metodiky jednotlivých testů.

6.1 Příprava vývojového prostředí

Před začátkem implementace je nutné nainstalovat testovací knihovny a provést konfiguraci aplikace pro testování. Technologie pro testování byly zvoleny v analytické části. Balíčky *phpunit/phpunit* a *symfony/panther* byly nainstalovány pomocí composeru.

6.1.1 Konfigurace docker kontejnerů

Programátoři v Blueghostu využívají k vývoji Docker prostředí spravované pomocí Docker Compose. Připomeňme si kontejnery platformy z úvodní kapitoly. Důležitý je kontejner **builder**, ve kterém programátoři spouštějí příkazy. Testy pro své fungování vyžadují instanci prohlížeče, za kterou byl v analytické části zvolený **chromedriver**.

Instalace chromedriveru

První vyzkoušenou variantou byla instalace chromedriveru přímo do kontejneru builder. Přímá instalace je jednoduché řešení pro zpřístupnění chromedriveru v docker kontejneru. Nevýhodou je, že v kontejneru bez GUI (grafické uživatelské rozhraní) se chrome spouští v tzv. headless režimu, ve kterém programátor nemůže sledovat průběh vykonávání testů. To je zásadní překážkou při lazení chyb během vývoje testů.

Druhá varianta je vytvoření nového kontejneru s operačním systémem obsahující GUI. Sledování průběhu testů je možné skrze VNC[1] server, který umožňuje připojení do kontejneru z vnější. To obnáší velké množství konfigurace s přípravou kontejneru a správným nastavením chromedriveru. Problémové je výchozí chování session cookies, které se uchovávají napříč jednotlivými testy.

[1] Virtual Network Computing - Protokol pro vzdálený přístup k grafickému uživatelskému rozhraní

Požadovaný stav je, aby po každém testu došlo k jejich vymazání a testy tak probíhaly v čisté instanci prohlížeče.

Finální varianta je použití připraveného Docker image *selenium/standalone-chrome*[2]. Ten přichází s již připravenou konfigurací pro připojování z vnější. Pro každý test je založena nová instance prohlížeče. Dále obsahuje Selenium Grid, který programátorům poskytuje jednoduchý přehled průběhu testů a stavu instancí prohlížeče.

Testovací prostředí

Symfony při spouštění aplikace rozlišuje tři základní prostředí, kterými jsou *prod*, *dev* a *test*. Každé prostředí načítá proměnné ze zvláštního souboru. Proměnné prostředí umožňují zvolit používanou databázi a další nastavení týkající se konkrétního prostředí. Použití jiné databáze pro testovací prostředí je důležité, aby programátorovo vývojová data nekolidovaly s daty při testování.

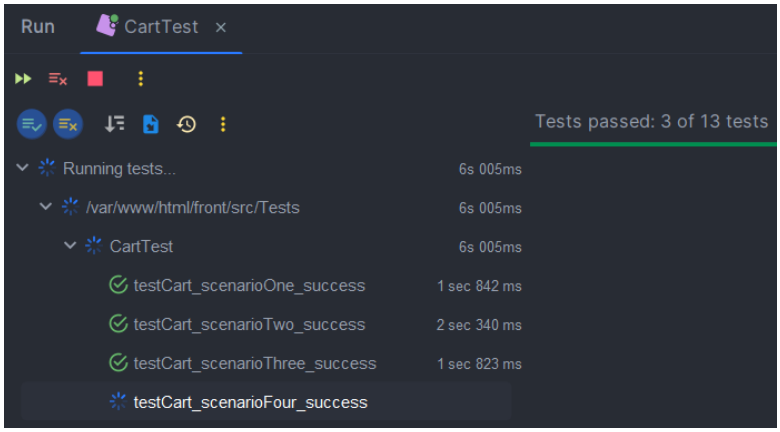
Prohlížeč v selenium gridu k aplikaci přistupuje z vnější. Nemá tak možnost aplikaci předat informaci o požadovaném prostředí, které je ve výchozí hodnotě nastavené na *dev*. Docker compose umožňuje spouštění kontejnerů skrze více konfiguračních souborů. Pro účely testování byl založen nový soubor *docker-compose.test.yml*, ve kterém jsou definovány dva zcela nové kontejnery pro backend a frontend aplikace. Oba kontejnery mají prostředí definované na *dev*. Testovací aplikace tak běží zcela odděleně od běžné vývojové. Výhodou takového přístupu je, že při spouštění Docker compose lze zvolit, jaké konfigurační soubory se mají použít. Pokud programátor zrovna nepotřebuje testovat, nemusí tyto kontejnery spouštět a může tím šetřit výkon svého počítače.

6.1.2 Nastavení PHPStorm pro spouštění testů

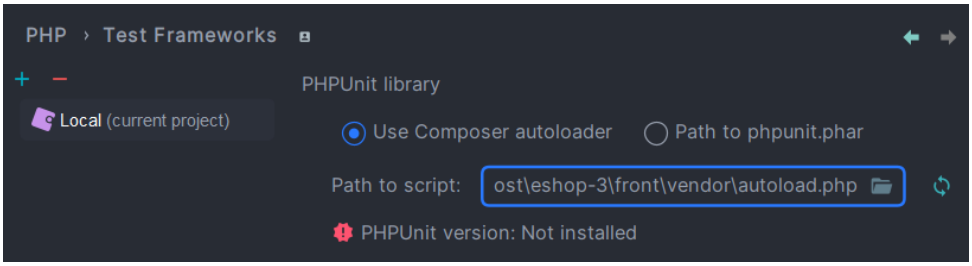
Vývojový nástroj PHPStorm obsahuje **test runner**. Jedná se o nástroj, který slouží ke spouštění testů. Díky vypracovanému UI se jedná o příjemnou variantu oproti spouštění testů pomocí příkazové řádky. Protože aplikace běží uvnitř docker kontejnerů, je potřeba PHPStorm nastavit, aby se do kontejnerů uměl připojit a testy spustit.

Nastavení se provádí sekci Command line, kde je potřeba založit nový interpreter. Zde se založí nový server Docker Compose server s daemonem *Docker for Windows*. Lifecylce se používá *docker-compose exec*. Protože se projekt skládá ze dvou aplikací, je potřeba v nastavení PHPStormu zvolit správnou cestu pro autoloader. Nastavení se provádí v sekci *PHP > Test Frameworks* a složka musí odpovídat aplikaci, ve které se mají spouštět testy. Pro frontend se zvolí složka front a pro backend back.

[2] `https://hub.docker.com/r/selenium/standalone-chrome`



Obrázek 6.1: Výpis nástroje test runner.



Obrázek 6.2: Nastavení autoloaderu pro PHPUnit.

6.2 Testování API endpointů

Testované endpointy a jejich hodnoty pro testování byly určeny v analytické části. Testy se týkají pouze API a proto jsou implementovány v backendové aplikaci.

6.2.1 Struktura souborů

Struktura testů backendové aplikace vypadá následovně.

```
tests/
├── data/ ..... Složka s daty potřebnými pro testování
├── DataModel/ ..... Datové objekty použité v testech
│   └── OrderApiTestData
├── Order/ ..... Testy validace polí objednávky
│   ├── OrderApiTest ..... Abstraktní třída testů polí objednávky
│   ├── OrderCustomerApiTest ..... Testy osobních údajů
│   ├── OrderDeliveryApiTest ..... Testy kombinací dopravy
│   └── OrderEmailApiTest ..... Testy emailových adres
└── CartApiTest ..... Testy košíku
```

6.2.2 Fixtures testovacích dat

Před spuštěním každého testu dochází k přípravě testovacích dat pomocí Symfony Fixtures. Pro spouštění Fixtures je používána knihovna *liip/test-fixtures-bundle*, která umožňuje spouštění pouze vybraných Fixtures. Není tak potřeba spouštět všechny Fixtures projektu, nýbrž pouze vybrané Fixtures pro konkrétní test. Ke spouštění Fixutures dochází v metodě *setUp*, která je spouštěna před každým testem.

```
public function setUp(): void
{
    $this->client = self::createClient();
    $databaseTool = self::getContainer()
        ->get( id: DatabaseToolCollection::class)
        ->get();
    $databaseTool->loadFixtures(['App\DataFixtures\TestSessionFixture'], true);
}
```

Obrázek 6.3: Příklad použití Fixtures v testu.

Pro účely testování byla vytvořena *TestSessionFixuture* pro zakládání session tokenů. Backendová aplikace se na základě session tokenu rozhoduje, se kterými daty bude daný dotaz na API pracovat. Fixture založí tři tokeny s různými obsahy košíku. Tyto tokeny jsou využívány při testech manipulace obsahem košíku a při testech objednávkového formuláře.

6.2.3 Testování API manipulace obsahem košíku

Endpoint pro úpravu obsahu košíku je volán HTTP metodou PUT. Jako payload přijímá JSON, který obsahuje informace o produktu a operaci, která se s produktem má provést. Možné operace jsou *set*, *sub* a *add*. Dle zvolené operace se cílové počet kusů produktů nastaví, sníží nebo zvýší. Testované scénáře byly stanoveny v analytické části.

Testovací třída **CartApiTest** vychází ze Symfony *WebTestCase*, který poskytuje klienta pro odesílání HTTP dotazů. V této třídě jsou definovány všechny testy Všechny pro manipulaci obsahem košíku. Testy využívají metodu **createRequestAttributes**, která připraví data pro odeslání dotazu na základě session tokenu, názvu produktu, operaci a počtu kusů. Testy následně ověří návratovou hodnotu dotazu a ověří, že vrácená odpověď je korektní JSON. Vrácená odpověď ve formátu JSON se porovnává s očekávanými JSON odpověďmi, které jsou uložené ve složce **data**.

6.2.4 Testování API objednávkového formuláře

Objednávkový formulář je obsluhován pomocí dvou API endpointů, které vracejí odpovědi ve formátu JSON. První endpoint slouží k úpravě formulářových polí. Pole mohou být upravována hromadně i jednotlivě. Během

úprav pole nejsou validovány. K validaci dochází až při odeslání HTTP dotazu PUT na druhý endpoint. Druhý endpoint zvaliduje uložená pole a vytvoří objednávku. V případě neuspěchů vypíše neúspěšná pole.

Objednávkový formulář se skládá z velkého množství polí. Pro větší přehlednost byly testy rozděleny do několika tříd, které vycházejí z abstraktní třídy **OrderApiTest**. Abstraktní třída obsahuje metody pro odeslání dotazů na oba endpointy. Definuje testovací metodu, která přijímá objekt *OrderApiTestData*. V objektu jsou předávány hodnoty pro odeslání dotazů i očekávané odpovědi z endpointů. Třída obsahuje abstraktní funkci *orderApiDataProvider*, která poskytuje testovací data a je definována v jejích potomcích. Při testování je využíváno toho, že endpoint pro potvrzení objednávky vrací všechna neúspěšná pole. Testy úspěšných polí kontrolují pouze nepřítomnost chyby pro dané pole, což vede k menší složitosti testů.

Ve třídě **OrderEmailApiTest** je testováno pět validních a devět nevalidních emailů. Testy jsou zaměřeny na správný formát emailové adresy a na maximální délky její částí. Třída **OrderCustomerApiTest** testuje údaje o zákazníkovy. Testuje se pouze přítomnost chybových hlášek, jelikož aplikace očekává pouze vyplněnost těchto polí a nemá další požadavky na jejich obsah. Různé kombinace dopravy a plateb jsou testovány ve třídě **OrderDeliveryApiTest**. Zde jsou testovány všechny validní i nevalidní kombinace pro Českou Republika a Slovensko.

6.3 End-to-end testování

Testy pracující s prohlížečem byly implementovány ve frontendové aplikaci. Jejich cílem je ověřit funkčnost celého internetového obchodu skrze simulaci skutečného chování uživatele.

6.3.1 Struktura souborů

Struktura testů frontendové aplikace vypadá následovně. Složka PageObject obsahuje třídy všech stránek používaných při testování.

```
tests/
├── data/ ..................... Složka s daty potřebnými pro testování
├── DataModel/ ..................... Datové objekty použité v testech
│   ├── OrderTestData
│   └── TestProduct
├── FormData/ ..................... Objekty s formulářovými daty
│   ├── AddressData
│   ├── CompanyData
│   ├── OrderData
│   └── RegistrationData
├── PageObject/
│   ├── PageObject ..................... Abstraktní třída stránek
│   └── další potomci PageObject
└── Utils/
```

```
    └── OrderTestsReader ..... Čtení dat testů objednávky
├── CartItemsTest ..... Testy úpravy počtu kusů v košíku
├── CartTest ..... Testy práce s košíkem
├── MyTestCase ..... Abstraktní třída pro testování
├── OrderTest ..... Testy objednávek
└── RegistrationTest ..... Testy registrace
```

6.3.2 Skript pro přípravu databáze

Během spouštění testů dochází k vytváření velkého množství objednávek a uživatelských účtů. Spravování těchto dat pomocí fixtures by vyžadovalo velké množství kódu, který by musel být udržován. Namísto toho bylo využito exportu databáze, který již v aplikaci existuje. Export programátoři využívají během inicializace databáze testovacími daty pro vývoj. Použití exportu je ideální přístup přípravy testovacích dat, jelikož se aktualizuje s každou úpravou aplikace a nepřidělává se tak práce s údržbou fixtures.

Export databáze je spouštěn pomocí skriptu **init_test_db.sh**. Skript skrze příkazy nejdříve smaže starou databázi. Pomocí exportu založí novou a spustí nad ní migrace. Následně importuje jazykovou sadu. Celý běh skriptu trvá přibližně pět sekund. Spouštění skriptu je zařízeno skrze konfiguraci PHPUnit. Skript je pouštěn pouze při inicializaci PHPUnit, nedochází proto ke zbytečnému zpomalení běhu testů.

6.3.3 Třídy pro ovládání stránek

Řízení testů skrze prohlížeč vyžaduje velké množství operací. Každá interakce s prvkem stránky musí prvek nejdříve lokalizovat a případně počkat na jeho načtení. Aby se testy udržely přehledné využívají se třídy stránek. Ty poskytují metody k ovládání stránek. Takové metody mohou například vyplňovat formulář, klikat na tlačítka a celkově manipulovat s webovým prohlížečem. Samotný kód testů poté neřeší jak se stránkou interagovat, nýbrž pouze jakou akci chce provést.

Pro každou testovanou stránku byla založena její třída. Všechny třídy vycházejí z abstraktní třídy **PageObject**. Ta se ve svém konstruktoru stará o to, že požadovaná stránka byla skutečně načtena. Dále obsahuje metody pro ovládání prvků, které se vyskytují na všech stránkách obchodu. Tyto prvky jsou například tlačítka pro volbu jazyka a měny, nebo ikonka košíku, ze které lze přečíst současnou hodnotu nákupu.

Lokalizace prvků webu

Při lokalizaci prvků je nutné předcházet možným změnám ve struktuře stránky. Není dobré vybírat prvky na základě jejich polohy ve stromové struktuře stránky. Jakýkoli zásah kodéra může strukturu změnit a tím testy zcela rozbít. Selektory prvků by měly být co nejjednoduší.

Ve spolupráci s kodérem byly všechny prvky potřebné pro testování rozšířeny vlastní třídou. Aby se CSS třídy nepletly s třídami určenými pro stylovaní,

začínají předponou **tst-**. Vyhledávání prvků na základě přiřazené třídy předchází budoucím problémům při úpravách struktury stránek.

Klikání na prvky mimo zobrazenou část stránky

Webový emulátor nedokáže kliknout na prvek stránky, který se nachází mimo obrazovku. Před každým kliknutím na prvek je potřeba stránku posunout tak, aby se prvek stal viditelný. Tato funkcionalita byla implementována v abstraktní třídě PageObject v metodě **selectCheckbox** pomocí Javascriptu. Metoda se nejdříve zeptá emulátoru na polohu prvku. Následně v emulátoru spustí Javascript, který stránku posune na požadované souřadnice.

Čekání na prvky načítané AJAXem

Automatické testy na rozdíl od člověka zvládají stránku ovládat téměř okamžitě. Vyplnění komplexního formuláře je otázka řádově několika milisekund. Často se tak stane testy probíhají příliš rychle a stránka se nestíhá načítat. Například rozbalování podformulářů nebo průběh animace mohou trvat stovky milisekund. Tyto prodlevy se však nasčítají a průběh celého testu může být o několik sekund pomalejší. Aby se předcházelo zbytečnému čekání, je využíváno metody **waitForVisibility**. Využitím metody se optimalizuje doba čekání na nejmenší potřebnou.

Další překážkou běhu testů jsou události stránky řízené AJAXem. Zpracování AJAX dotazu trvá podstatně déle a může měnit obsah stránky. Pokud testy pracují s prvkem stránky, který je při běhu testů nahrazen, či zcela odebrán, dojde k chybě *Stale Element Exception*. Tato chyba značí, že prvek není součástí stránky a již není možné s ním dále pracovat. Při interakci se stránkou je potřeba brát ohled na to, které operace vyvolají změnu obsahu stránky. U všech měněných prvků je potřeba počkat na stav **WebDriverExpectedCondition::stalenessOf**. Tento stav zaručí, že původní prvek byl ze stránky odebrán a se stránkou je možné dále pracovat.

6.3.4 Testování tvorby objednávek

Odesílání objednávek je nezbytné pro správné fungování internetového obchodu. Z testů API endpointů již je ověřeno, že dochází ke správné validaci jednotlivých polí. Cílem těchto testů je ověřit, že dojde ke správnému založení objednávky na základě různých kombinací, které byly stanoveny v analytické části.

Testování probíhá ve třídě **OrderTest**. Kombinace vstupních hodnot jsou generovány providerem, který vrací objekty typu **OrderTestData**. Tento objekt obsahuje všechna data potřebná k otestování jedné objednávky. Provider generuje testovací data pomocí třídy **OrderTestsReader**, která načítá data ze souboru *orderTestData.csv* ve složce data. Načítání ze souboru bylo využito z důvodu velkého množství vstupních kombinací, jejichž definice v kódu by byla neudržitelná.

V průběhu každého testu objednávky dojde k naplnění košíku na základě testovacích produktů v objektu **OrderTestData**. Následně se vyplní objednávkový formulář dle objektu **OrderData**. Po odeslání objednávky je prohlížeč přesměrován na stránku o jejím úspěšném odeslání, kde se kontroluje přítomnost hlášky *Objednávka byla úspěšně odeslána*. Posledním krokem testování je návštěva detailu objednávky. Zde se zkontrolují počty produktů, zvolená doprava a všechna další pole odeslané objednávky.

6.3.5 Testování registrace

Registrace je testována ve třídě **RegistrationTest**. Registrační formulář obsahuje pouze čtyři pole, definice testovacích dat jsou proto umístěny přímo v provideru. Při testování dojde k odeslání registračního formuláře a následně je kontrolována přítomnost hlášky *Registrace proběhla úspěšně*, nebo očekávané chyby.

6.3.6 Testování manipulace obsahem košíku

Testy manipulace obsahem košíku jsou rozděleny do dvou tříd. Ve třídě **CartTest** jsou implementovány testy, které se zaměřují na úpravy košíku napříč různými stránkami internetového obchodu. Testovací scénáře byly definovány v analytické části. Třída obsahuje všech 17 scénářů. Každý krok v testovacím scénáři představuje jedno zavolání metody objektu příslušné stránky. Po každém kroku je kontrolována celková cena nákupu.

Testy zaměřené na úpravu obsahu košíku ze stránky pro vytvoření objednávky jsou implementovány ve třídě **CartItemsTest**. Obdobně jako v předchozí třídě, je při každém kroku ve scénáři zavolána meta objektu stránky. Po každém kroku je kontrolována celková cena nákupu a počty kusů každého produktu.

Kapitola 7

Závěr

7.1 Shrnutí

V bakalářské práci byla definována pravděpodobnost selhání a možné poškození pro každou komponentu platformy. Výsledky byly konzultovány s technickým ředitelem agentury BlueGhost a na jejich základě byly určeny komponenty, pro které byl vypracován plán testů. Komponenty zvolené pro testování jsou *košík, objednávka a registrace*.

Byla provedena rešerše testovacích technologií vhodných pro použití v Symfony frameworku, kterými jsou *PHPUnit, Symfony WebTestCase, Panther a Selenium*.

Do platformy pro tvorbu internetových obchodů byly pomocí Docker compose přidány kontejnery pro testovací prostředí aplikace. Prostředí běží zcela odděleně od zbytku aplikace, a tak programátorům umožňuje průběžné testování během vývoje. V backendové aplikaci byly vytvořeny Fixtures pro přípravu testovacích dat. Pro systémové testy frontendové aplikace, které vyžadují větší množství upravených dat, byl vytvořen skript pro přípravu testovací databáze.

Dle plánu testů byly implementovány systémové testy frontendové aplikace. Celkem bylo implementováno 17 testů pro manipulaci s obsahem košíku, 58 testů objednávkového formuláře a 7 testů registračního formuláře. Při implementaci testů bylo využito návrhového vzoru PageObject a byl kladen důraz na budoucí udržitelnost testů. Dále byly implementovány API testy pro backendovou aplikaci, které obsahují 28 testů pro validaci polí objednávkového formuláře a 11 testů úpravy obsahu košíku.

Hezký příklad účinnosti navrženého a implementovaného testování platformy je odhalení chyby v produkčním prostředí. Ačkoli platforma běží v provozu několik let a měla by být řádně otestována, během implementace testů **byla nalezena chyba** na detailu objednávky. K chybě došlo, pokud objednávka obsahovala hrazenou dopravu. Následkem chyby nebylo možné zobrazit detail objednávky. Jednalo se přesně o příklad chyby, **kvůli které byly testy implementovány**. Chyba byla způsobena zásahem programátora do vykreslování produktů v objednávce. Při následném testování neodhalil speciální případ, kdy součástí objednávky je placená doprava. Opomenul tak jednu z mnoha kombinací, které jsou nyní pokryty automatickými testy.

7.2 Manuál tvorby testů

V BlueGhostu jsou veškeré postupy dokumentovány ve firemní wiki. V rámci práce byla vytvořena wiki stránka obsahující manuál pro tvorbu automatizovaných testů. Manuál popisuje potřebné nastavení vývojového nástroje PHPStorm, doporučené strukturování testů a problémy, které bylo potřeba vyřešit v průběhu implementace.

7.3 Vyzkoušení výsledků práce

Implementovaný kód byl odevzdán společně s bakalářskou prací. Samotný kód testů ovšem nelze spustit bez testované aplikace, kterou není možné zveřejnit. Ukázka běhu testů je možná ve firmě BlueGhost po osobní domluvě na jednom z emailů:

- jan.vales@blueghost.cz
- michal.zima@blueghost.cz
- info@blueghost.cz

Literatura

[1] Certifikovaný tester základní úrovně, International Software Testing Qualifications Board, Verze 2018 v3.1 [online]. [Cit. 12. 1. 2023] Dostupné z: https://castb.org/ke-stazeni

[2] Patton, R. Testování softwaru. Computer Press, 2002. ISBN 80-7226-636-5.

[3] Bureš, M. et al. Efektivní testování softwaru: Klíčové Otázky pro Efektivitu Testovacího Procesu. Grada, 2016. ISBN 978-80-247-5594-6.

[4] Testing (Symfony Docs) [online]. [Cit. 27.9.2022] Dostupné z: https://symfony.com/doc/current/testing.html

[5] PHP: What is PHP? [online]. [Cit. 12. 1. 2023] Dostupné z: https://www.php.net/manual/en/intro-whatis.php

[6] Getting Started with Doctrine [online]. [Cit. 12. 1. 2023] Dostupné z: https://www.doctrine-project.org/projects/doctrine-orm/en/2.15/index.html

[7] Managing CSS and JavaScript [online] [Cit. 12. 1. 2023] Dostupné z: https://symfony.com/doc/current/frontend.html

[8] jQuery [online]. [Cit. 12. 1. 2023] Dostupné z: https://jquery.com

[9] Docker overview [online]. [Cit. 12. 5. 2023] Dostupné z: https://docs.docker.com/get-started/overview/

[10] Introduction to Selenium Automation Testing [online]. [Cit. 19. 5. 2023] Dostupné z: https://www.guru99.com/introduction-to-selenium.html

[11] What is Selenium Grid? [online]. [Cit. 19. 5. 2023] Dostupné z: https://www.javatpoint.com/selenium-grid

Příloha A

Manuál tvorby testů

Automatizace testování

Jednou z funkcí Eshop3.1 jsou automatizované testy. Testy nejsou implementovány do CI/CD procesu, musejí se spouštět lokálně. Každý programátor by testy měl spouštět **po ukončení vývoje feature**.

Obsah

Spouštění docker-compose kontejnerů

Aplikace testovacího prostředí běží ve vlastních kontejnerech s vlastní databází **eshop3_test**. Databáze je automaticky založena při spouštění frontendových testů, nebo skriptem `./init_test_db.sh`. Testovací kontejnery jsou definovány ve vlastním souboru **docker-compose.test.yml**, aby nebyly vždy spuštěny a šetřil se výkon počítače. Spustit testovací kontejnery lze pomocí `docker-compose -f docker-compose.yml -f docker-compose.test.yml up`.

Testovací kontejnery jsou dostupné na adrese **eshop3.test** (potřeba přidat do hosts viz readme).

Spouštění testů v příkazové řádce

Testy se spouštějí pomocí phpunit nainstalované v backendové i frontendové aplikaci.

Spuštění všech testů.

```
php bin/phpunit
```

Spuštění testů dle názvu testovací metody.

`php bin/phpunit --filter="nazev"`, kde název je část názvu testu, například "order".

Pokud jsou data poskytována data providerem, lze uvést pořadové číslo.

`php bin/phpunit --filter="#0"`, kde 0 je první iterace všech testů řízených data providerem.

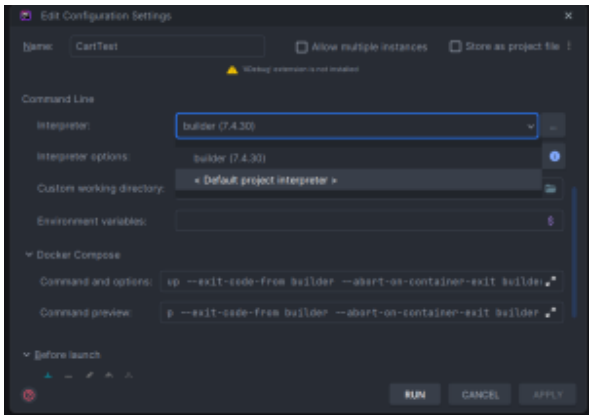
Filtry lze i kombinovat.

```
php bin/phpunit --filter="order#0"
```

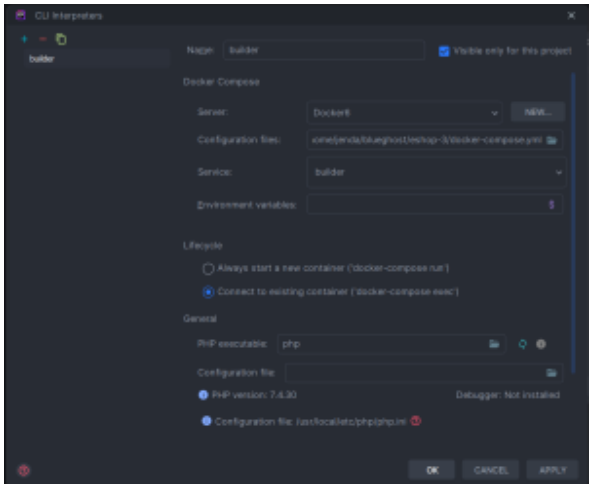
Spouštění testů v PHPStorm

PHPStorm umí spouštět testy pomocí test runneru. Protože front i back mají vlastní nastavení PHPUnit, je nutné mít při testování otevřený konkrétní projekt front/back.

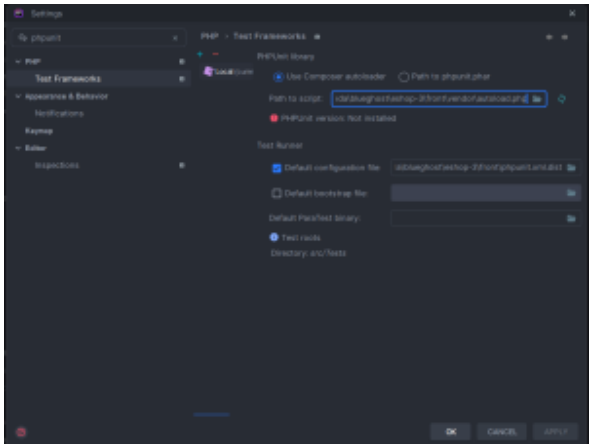
Pro nastavení testů založte nový interpreter.



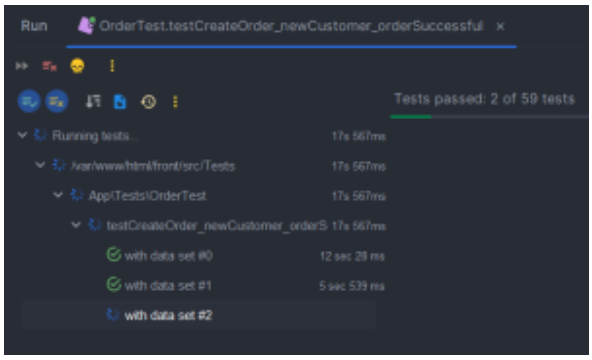
V interpreteru nový server s daemonem docker for windows.



Dále je v PHPStormu upravit v nastavení PHPUnit cestu k autoloadu a default configuration file.



Výsledkem jsou testy spustitelné pomocí přehledného UI PHPStormu.



Doporučení k implementaci nových testů

Vhodná šabolna pro pojmenování testovacích metod je test{čeho}_{počáteční stav}_{koncový stav}. Například **testBasketSet_emptyBasket_itemAdded**.

PHPUnit podporuje fixtures metody (https://dyclassroom.com/phpunit/phpunit-fixtures-setup-and-teardown) (něco jiného než data fixtures v Symfony). Vhodná je zejména metoda **setUp()**, která je volána před každým testem.

K testování více kombinací pomocí jediné metody slouží **DataProvider** anotace. Použití je následující:

```
/**
* @dataProvider sumDataProvider
*/
public function testSum_integersGiven_integerReturned(int $first, int $second, int $expectedResult): void
{
    ....
}

public function sumDataProvider(): \Generator
{
   //Hodnoty lze vracet jednotlivě, "Jedna plus dva" je název datasetu zobrazený při neúspěšném testu
   yield "Jedna plus dva" => [1, 2, 3];

   //Nebo v libovolném cyklu
   for($i = 0; $i < 100; $i++) {
       yield [$i, 0, $i];
```

```
    }
}
```

Backendové API testy

Testování API endpointů je implementováno v backendové aplikaci. K API je přistupováno pomocí klienta, který je součástí Symfony třídy **WebTestCase**.

Přípravu testovacích dat je vhodné provádět pomocí **fixtures**, jejichž volání lze jednoduše provést v setUp funkci.

```
public function setUp(): void

{

$this->client = self::createClient();

$databaseTool = self::getContainer()->get(DatabaseToolCollection::class)->get();

$databaseTool->loadFixtures(['App\DataFixtures\TestSessionFixture'], true);

}
```

Frontendové systémové testy

K testování frontendové aplikace je využíváno emulátoru prohlížeče, který běží uvnitř vlastního kontejneru. Přístupný je na adrese **selenium.local**.

Vytvoření instance prohlížeče je implementováno v abstraktní třídě **MyTestCase** v metodě **createSeleniumClient**. Metoda založí Selenium client, se kterým má samotný Panther problém.

Testy frontendové aplikace pracují s větším množstvím dat, které by bylo náročné udržovat pomocí fixtures. PHPUnit je proto nastavená, aby při každém spuštění testů přenačetla databázi testovacími daty (stejné jako spuštění `./install.sh`).

Objekty stránek

Během implementace testů je potřebné interagovat s konkrétnimi elementy stránky (například tlačítko přidat produkt na detailu produktu). Aby se kód neopakoval napříč různými testy, zakládají se "PageObjects (https://www.browserstack.com/guide/page-object-model-in-selenium#:~:text=Factory%20in%20Selenium-,What%20is%20Page%20Object%20Model%20in%20Selenium%3F,and%20improves%20test%20case%20maintenance.)". Ty poskytují programátorovi interface pro ovládání stránky. Metody implementované v objektu domovské stránky mohou být například `gotoLogin(): LoginPage`, `gotoProduct(string $name): ProductDetailPage` a `setProductCount(string $name, int $count): void`.

Ideální nástroje pro výběr elementů jsou CSS selektory a XPath. Selektory by měly být co nejvíce **jednoduché a jednoznačné**, aby se předcházelo problémům při kodérských úpravách.

Všechny CSS selektory musejí mít předponu **tst-**, podobně jako pro javascript používáme *js-*.

Čekání na načtení

Při čekání na načtení stránky, nebo na jinou událost je **nevhodné používat sleep()!** Ideální je použít jednu z metod **waitFor**, **waitForVisibility**, **waitForInvisibility**, **waitForAttributeToContain**, které čekají **nejmenší potřebný čas.**

Pozor na AJAX! Pokud AJAX provede úpravu nebo zcela odstraní element, se kterými testy pracují, dojde k *StaleElementReferenceException*. Staleness je stav elementu, který již není součástí DOM. K počkání na dokončení AJAXu lze použít `WebDriverExpectedCondition::stalenessOf($element)`, kde argument je element, který po dokončení AJAXu nemá být součástí DOM.

Stránka byla naposledy editována 21. 5. 2023 v 23:36.