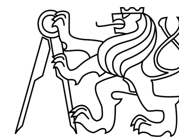
ESKÉ VYSOKÉ U ENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMA NÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁ SKÉ PRÁCE

Název: Automatizované testování webového portálu dbs.fit.cvut.cz
Student: Pavel Ková
Vedoucí: Ing. Ji í Hunka
Studijní program: Informatika
Studijní obor: Softwarové inženýrství
Katedra: Katedra softwarového inženýrství
Platnost zadání: Do konce letního semestru 2017/18

Pokyny pro vypracování

Pro p edm t BI-DBS vzniká v rámci projekt BI-SP1 a BI-SP2 nástroj pro celkovou podporu výuky - portál dbs.fit.cvut.cz. B hem vývoje jsou nové funkcionality manuáln testovány na vybraných vstupech. Nanešt stí, tento p ístup neodhalí chyby, které mohou vzniknout v ostatních ástech aplikace, a do produkce se tak dostává i nevalidní kód.
Cílem této práce je navrhnout a realizovat ešení pro automatické testování aplikace tak, aby vzniklé chyby byly odhaleny nejen p i nasazení aplikace do produk ního prost edí, ale i v pr b hu jejího vývoje.

1. Prostudujte možnosti testování p i vývoji aplikací realizovaných v obdobných technologiích.
2. Navrhn te vhodné metody a postupy pro testování p i vývoji portálu.
3. Navržené metody ov te na aktuální verzi portálu.
4. Zhodno te výsledky testování.
5. Zprovozn te pr b žné testování využitelné i p i automatickém nasazení.
6. Vytvo te dokumentaci nebo podp rné materiály pro budoucí vývojá e projektu.

Seznam odborné literatury

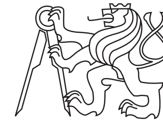
Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
d kan

V Praze dne 22. prosince 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového inženýrství



Bakalářská práce

Automatizované testování webového portálu dbs.fit.cvut.cz

Pavel Kovář

Vedoucí práce: Ing. Jiří Hunka

15. května 2017

Poděkování

Chtěl bych poděkovat panu Ing. Jiřímu Hunkovi za možnost zvolit si toto téma jako svojí bakalářskou práci. Dále bych chtěl poděkovat Oldřichu Malcovi za vynikající vedení týmů v rámci předmětů BI-SP1 a BI-SP2. Mé poděkování patří též mé rodině za jejich podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2017 ....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2017 Pavel Kovář. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Kovář, Pavel. *Automatizované testování webového portálu dbs.fit.cvut.cz.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato bakalářská práce se zabývá testováním webového portálu vyvíjeného pro podporu výuky předmětu Databázové systémy na Fakultě informačních technologií ČVUT v Praze. Cílem práce je navrhnout a vytvořit prostředí vhodné pro automatizované testování webového portálu. Práce se zabývá jednak stanovením vhodných prostupů pro testování webových aplikací a jejich ověřením na současné vývojové verzi portálu, ale i návrhem prostředí využitelného pro průběžné testování. V práci je mimo jiné zmíněn vývoj několika modulů pro testovací framework Codeception, díky nímž je možné testovat také aplikace vytvořené v Nette frameworku.

Klíčová slova softwarové testování, databázové systémy, webový portál, průběžné testování, Nette, Codeception

Abstract

This bachelor thesis deals with the testing of a web portal developed to support the teaching of Database Systems subject at the Faculty of Information Technology, CTU in Prague. The aim is to design and create an environment suitable for automated testing of the developed web portal. The thesis deals with the choice of suitable testing techniques for testing web applications and their validation on the current development version of the portal, as well as with the design of environment usable for continuous testing. The work also mentions the development of several modules for the Codeception testing framework, which allows to test web applications created with Nette framework.

Keywords software testing, database systems, website, continuous testing, Nette, Codeception

Obsah

Úvod 1

1 Základy testování software 3
- 1.1 Zavedení pojmů . . . 3
- 1.2 Příprava testování . . . 3
- 1.3 Přístupy k testování . . . 4
- 1.4 Statické testování . . . 6
- 1.5 Dynamické testování . . . 7
- 1.6 Pravidla pro tvorbu testů . . . 11
- 1.7 Shrnutí . . . 12

2 Systém pro podporu předmětu BI-DBS 13
- 2.1 Předmět Databázové systémy . . . 13
- 2.2 Vznik a vývoj . . . 14
- 2.3 Architektura a použité technologie . . . 14
- 2.4 Funkcionalita systému . . . 17
- 2.5 Stav testování . . . 19

3 Návrh 21
- 3.1 Testovací prostředí . . . 21
- 3.2 Testovací techniky . . . 22
- 3.3 Průběžné testování . . . 23
- 3.4 Automatické nasazení . . . 24

4 Realizace 25
- 4.1 Tvorba prostředí . . . 25
- 4.2 Vývoj podpůrných nástrojů a modulů . . . 27
- 4.3 Průběžné testování . . . 29
- 4.4 Automatické nasazení . . . 31
- 4.5 Zhodnocení testování . . . 32

4.6 Dokumentace 33

Závěr **35**

Literatura **37**

A Seznam použitých zkratek **41**

B Kodovací standard **43**

B.1 Hlavní pravidla 43

B.2 Další pravidla 43

C Obrázky **47**

D Obsah přiloženého CD **49**

Seznam obrázků

2.1 Diagram nasazení systému v produkčním prostředí včetně vazeb na externí systémy . . . 16

3.1 Předpokládaný průběh testování při vytvoření nové revize v repozitáři projektu . . . 23
3.2 Předpokládaný proces nasazení . . . 24

4.1 Vizualizace testovacího prostředí vytvořeného v nástroji Docker . . 26

C.1 Proces průběžného testování . . . 48

Seznam tabulek

2.1 Parametry serveru 17

4.1 Statistika nahlášených chyb a návrhů na zlepšení 32

Úvod

Testování, jedna z nejdůležitějších, avšak nejčastěji opomíjených částí vývojového cyklu jakéhokoliv software. I přes naši snahu tvořit bezchybný kód se do vytvářené aplikace dostávají chyby. Čím více je výsledná aplikace složitější, tím vyšší je pravděpodobnost výskytu ať už kritických či méně závažných chyb. Přitom právě díky testování můžeme tyto chyby včas odhalit a předejít jejich eliminaci v pozdějších fázích vývoje, kdy je to pro nás v lepším případě časově a finančně nákladnější, v horším případě utrpí také naše reputace. Přece jen naším hlavním cílem by mělo být dodat zákazníkovi kvalitní software, se kterým bude spokojen.

Zejména přítomnost chyb trápí v současné době také vývoj webového portálu, jenž vzniká pro podporu předmětu Databázové systémy na Fakultě informačních technologií Českého vysokého technického učení v Praze. Portál si klade za cíl nejen umožnit studentům vytvářet semestrální práce a psát zápočtové i zkouškové testy online, ale umožnit také učitelům ohodnotit takto vytvořené testy či semestrální práce. Výsledný portál tedy značně automatizuje výuku celého předmětu.

Samotný portál je vyvíjen dvěma skupinami lidí. První skupinou jsou studentské týmy, které vznikají v rámci předmětů Softwarový týmový projekt 1 a Softwarový týmový projekt 2. Zbývající skupinou jsou studenti, kteří se na vývoji podílí v rámci své závěrečné práce. Naneštěstí během vývoje je značně podceněno testování, kvůli čemuž se čas od času objeví v produkční verzi portálu i naprosto zbytečné chyby.

V současné době lze k testování portálu využít sadu automatických testů, avšak tyto testy jsou závislé na okolním prostředí, což způsobuje, že je nelze po určité době spustit. Z toho důvodu je portál testován především manuálně a to tak, že vývojář, který vyvíjí novou funkcionalitu ji také otestuje na vstupech, které si sám zvolí. Bohužel takto vytvořený testovací případ již není nikde zaznamenán. Při dalším testování tak nedochází k ověření všech možných průchodů aplikací, a proto se čas od času stane, že se v produkčním prostředí objeví také chyby ve funkcionalitách, které dříve pracovali bez

jakýchkoliv chyb.

Cílem této práce je navrhnout a vytvořit prostředí vhodné pro testování vyvíjeného webového portálu. Práce se zabývá jednak stanovením vhodných postupů pro testování webových aplikací a jejich ověřením na současné verzi portálu, ale i automatizací procesu testovaní s ohledem na budoucí využití při automatickém nasazení.

Tato práce souvisí s bakalářskými pracemi Filipa Glazara [1] a Petra Pejši [2], které se věnovali počátečnímu vývoji testovaného portálu.

Kapitola 1

Základy testování software

Způsobů testování software existuje celá řada. Některé jsou vhodné pro testování jednoduchých aplikací, zatímco jiné jsou vhodné pro testování komplexních systémů. Všechny však mají shodný cíl – učinit software kvalitnějším a zákazníka spokojenějším.

V této kapitole se nejprve shrnu, co bychom měli mít připravené před testováním, následně rozeberu vybrané techniky testování software, a nakonec zmíním praktiky, které nám pomohou vytvářet lépe udržovatelné testy. Jelikož se tato práce zabývá testováním webového portálu, zaměřím se především na techniky, které lze využít při testování webových aplikací.

1.1 Zavedení pojmů

Než se podrobněji podíváme na jednotlivé testovací techniky, zavedu nejdříve pojmy *testovací případ* a *testovací scénář*, které budu dále v práci používat.

„Testovací případ popisuje konkrétní akce, prováděné s určitou softwarovou komponentou, a jejich očekávané výsledky.“ [3] Softwarovou komponentou může být v tomto případě například prvek uživatelského rozhraní nebo část aplikačního rozhraní systému.

Testovací scénář je složen z několika logicky po sobě jdoucích testovacích případů, jejichž účelem je otestovat vybranou funkcionalitu vyvíjeného systému. V kontextu systému, testovaného v rámci této práce, to může být například vytvoření semestrální práce studentem.

1.2 Příprava testování

Předtím než se vrhneme na testování, měli bychom nejdříve stanovit, nebo si alespoň pořádně rozmyslet, jakým způsobem budeme postupovat. Podle [4, 5] bychom si měli položit alespoň tři základní otázky týkající se přípravy testování.

1.2.1 Cíle testování

Dříve než začneme s testováním, měli bychom nejprve stanovit cíle testování a ujistit se, že jsou měřitelné. Každému cíli bychom měli stanovit také jeho prioritu dle požadavků a očekávání zákazníka. S takto vzniklým dokumentem nejprve seznámíme ostatní členy testovacího týmu a přesvědčíme se, že mu rozumí a dodržují jej. Nakonec jej přiložíme do projektové dokumentace.

1.2.2 Zpracování a hlášení chyb

Každý člen testovacího týmu musí znát svojí roli a vědět co, kdy, komu a kam má hlásit. Jinými slovy měli bychom specifikovat průběh testovacího procesu. Abychom mohli testovací proces lépe definovat, můžeme využít následujících otázek, jejichž zodpovězení nám při rozhodování pomůže.

- Jak a kam budou chyby hlášeny?
- Jak budou chyby kategorizovány?
- Kdo komu může chybu přiřadit?
- Kdo a kdy potřebuje něco hlásit?
- Jsou schůzky týmu naplánovány předem nebo když je to potřeba?

Testovací proces můžeme definovat formálně ale i neformálně, záleží na konkrétních potřebách našeho týmu. Hlavním smyslem je dosáhnout takové organizace týmu, která co nejvíce podpoří dosažení stanovených cílů. [4]

1.2.3 Testovací prostředí

Před zahájením testování bychom si měli připravit prostředí, ve kterém se bude testování vyvíjeného systému odehrávat. Vytvořené prostředí musí být nezávislé na produkčním i vývojovém prostředí, avšak musí být obdobou těchto prostředí, jinak bychom vyvíjený systém nemohli spustit, tudíž ani testovat.

Při návrhu prostředí bychom měli také definovat jakým způsobem bude k prostředí přistupováno, respektive jak bude probíhat nasazení nového aplikačního kódu. Kromě toho bychom měli také určit, jakým způsobem budeme schopni identifikovat právně nasazenou verzi systému.

1.3 Přístupy k testování

V této kapitole se nejprve zaměřím na nejčastější přístupy k návrhu testovacích případů, nakonec zmíním přístupy k vykonávání vytvořených testů.

1.3.1 Testování černé, bílé a šedé skříňky

Z pohledu viditelnosti zdrojového kódu lze rozdělit testování software do tří skupin:

Testování černé skříňky (black box)

Během testování černé skrínky si můžeme software představit jako černou skříňku, u které víme, jak se navenek chová, avšak nevíme nic o její vnitřní struktuře. Testování tedy probíhá tak, že na vstup předkládáme předpřipravená data a následně vyhodnocujeme, zda se na výstupu objevila očekávaná odpověď.

Testování bíle skříňky (white box)

Při testování bíle skřínky známe vnitřní strukturu testovaného systému, případně máme k dispozici také jeho zdrojový kód. Díky dokonalé znalosti vnitřní struktury můžeme snadno zjistit, jaká data musíme na vstup systému zadat, abychom otestovali všechny možné průchody systémem.

Testování šedé skříňky (gray box)

„Při testování šedé skříňky se částečně stírá hranice mezi testováním černé skříňky a bílé skříňky. Software testujeme jako černou skříňku, ale zároveň jako doplnění testů nahlížíme do mechanismů jeho činnosti (nenahlížíme do něj ale úplně jako u testování bílé skříňky).“ [6]

1.3.2 Testování splněním a selháním

Během návrhu testovacích případů bychom se měli zabývat nejen ověřením pozitivních průchodů systémem, ale také situacemi, které mohou způsobit nepředvídatelné chování systému. Abychom ověřili obě zmíněné situace, můžeme podle [7] využít následujících technik:

Testování splněním

Testování splněním si klade za cíl ověřit, zda se testovaný software chová dle dohodnuté specifikace požadavků (SRS). Naším cílem není systém „shodit“, tudíž během testování předkládáme systému pouze taková data, která musí za všech okolností akceptovat.

Testování selháním

Cílem testování selháním je vyvolat v systému takový stav, který způsobí nečekávané chování nebo pád celého systému. Při testování předkládáme na vstup hraniční či zcela nevalidní data a sledujeme, jak se s nimi systém vypořádá. Jestliže nám během testování systém poskytne nějaká chybová hlášení, pak bychom tato hlášení měli ověřit, zda odpovídají krokům, jenž provádíme.

1.3.3 Automatizované a manuální testování

Z pohledu, kdo testy vykonává, dělíme testování na automatizované a manuální. Na první pohled se může zdát, že automatizované testování musí být výhodnější, avšak vždy tomu tak není. Podívejme se podrobněji na oba přístupy a popišme si jejich výhody i nevýhody.

Manuální testování provádí skuteční lidé – testeři. Na rozdíl od stroje dokáže tester lépe reagovat na vzniklé situace nebo přesněji vyhodnotit slabá místa systému. Na druhou stranu výsledky od testera mohou být v některých případech subjektivně zkreslené nebo neúplné. Například pokud zmíněný tester zcela neporozumí problematice v testované části systému.

Automatizované testování, jak plyne z názvu, je prováděno strojově. Pro tento druh testování bychom se měli rozhodnout, pokud testování často opakujeme ale vykonávané testy zůstávají mezi testováními stále stejné. Jiným důvodem může být situace, kdy potřebujeme provést obrovské množství testů, které bychom při manuálním testování nebyli schopni v rozumném časovém horizontu vykonat. Nevýhodou tohoto řešení může být jednak vyšší časová náročnost na tvorbu nových testů, ale také nízká možnost analýzy vniklých chyb automatem. Chybný test musíme většinou vykonat také manuálně, abychom podrobněji identifikovali vzniklý problém a mohli jej vyřešit.

Přestože je automatizované testování stále populárnější, neměli bychom zapomínat, že automatizované nástroje nemohou skutečné testery v žádném případě nahradit, pouze jim pomáhají odvádět jejich práci snáze a lépe. [6]

1.4 Statické testování

Statické testování je specifické tím, že se neprovádí na spuštěném systému ba dokonce systém nemusí v danou chvíli vůbec existovat. Tento druh testování je vhodné aplikovat již v ranných fázích vývoje, kdy se zabýváme projektovou dokumentací či vytvářením prototypu výsledného systému. *„Typickým příkladem statického testování je testování jednotlivých dokumentů vzniklých v projektu a zdrojového kódu aplikace.“* [5]

1.4.1 Testování dokumentace

Při testování dokumentace se soustředíme především na kontrolu obsahu dokumentů z pohledu správnosti, úplnosti, jednoznačnosti a konzistence. Vzhledem k tomu, že testování dokumentů spadá do oblasti manuálního testování, nebudu se tímto druhem testování dále zabývat.

1.4.2 Testování zdrojového kódu

Zdrojový kód testujeme podobným způsobem jako jiné dokumenty nacházející se v projektu, avšak zde se zaměřujeme především na formu a obsah. *„V rámci*

kontroly obsahu se zaměřujeme na programátorské umění: jak robustní, odolný a efektivní kód programátoři dodávají. V rámci kontroly formy se zaměřujeme na přehlednost a čistotu kódu.“ [5]

Podobně jako u běžných textových dokumentů slouží ke stanovení určité formy šablony, u zdrojového kódu zastávají tuto roli kódovací standardy (angl. coding standards). Kódovací standardy tvoří sadu doporučení, které říkají, jak by měl být zdrojový kód napsán a strukturován. Příkladem může být doporučení říkající, jak bychom měli pojmenovávat proměnné, respektive jestli mají začínat malým či velkým písmenem.

Mezi metody statického testování zdrojového patří revize kódu (angl. code review), párové programování a automatizovaná analýza kódu. [5] Jelikož metody revize kódu a párového programování patří do oblasti manuálního testování, nebudu jimi více zabývat.

1.5 Dynamické testování

Dynamické testování se zabývá chováním spuštěného systému. Během testování se soustředíme na jednotlivé aspekty systému jako jsou spolehlivost, funkčnost, bezpečnost či uživatelská přívětivost, avšak naším hlavním cílem je ověřit, zda se testovaný systém chová dle požadavků a přání zákazníka.

Tento druh testování dále dělíme do několika skupin, podle toho, v jakém časovém horizontu od napsání zdrojového kódu jej provádíme.

1.5.1 Jednotkové testování

Jednotkové testy se zaměřují na testování systému z nejnižší úrovně. Testování probíhá obvykle na úrovni jedné funkce nebo třídy, z čehož je patrné, že se jedná o testování bílé skříňky. Cílem těchto testů je nejen důkladné prověření chování vymezeného bloku zdrojového kódu, ale také ověření nezávislosti bloku zdrojového kódu na okolním prostředí.

Pokud bychom chtěli zavést tento druh testování do již zaběhnutých projektů, můžeme zjistit, že to bude velice obtížné. Abychom tyto testy mohli aplikovat, musíme dosáhnout vzájemné nezávislosti bloků zdrojového kódu, což si může vyžádat hlubší zásahy do základů systému. V některých případech může jejich zavedení vyžadovat dokonce celkovou restrukturalizaci systému. Proto je vhodné se jednotkovým testováním zabývat již v počátečních fázích návrhu a vývoje systému.

1.5.2 Integrační testování

Integrační testy se zabývají vzájemnou interakcí komponent nejen uvnitř systému ale i s okolním prostředím. Podle [5] můžeme tyto testy rozdělit do dvou skupin:

Testování lokálního API

Testování probíhá na úrovni modulů uvnitř vyvíjeného systému. Jelikož vývoj všech modulů typicky neprovádí jedna skupina vývojářů, musíme se přesvědčit, že vytvořený modul je schopen komunikovat s ostatními částmi systému. Má-li být modul závislý jiné části systému, pak při testování ověřujeme, zda mezi těmito moduly dochází k bezchybné výměně informací. Naopak má-li modul pracovat nezávisle na okolí, pak sledujeme, zda za běhu modulu nedochází k narušení jiných částí systému.

Testování vzdáleného API

Testování se zabývá jednak rozhraním, které vyvíjený systém vystavuje svému okolí, ale také komunikací, která probíhá mezi rozhraními externích systémů a vyvíjeným systémem. Během testování zkoumáme, zda se systém chová stabilně i v případě, kdy se dostane do neočekávané situace. Například když obdrží neočekávaný požadavek, nebo získá nevalidní odpověď.

1.5.3 Systémové testování

„Během těchto testů je aplikace ověřována jako funkční celek. Tyto testy jsou používány v pozdějších fázích vývoje. Ověřují aplikaci z pohledu zákazníka. Podle připravených scénářů se simulují různé kroky, které v praxi mohou nastat. Obvykle probíhají v několika kolech. Nalezené chyby jsou opraveny a v dalších kolech jsou tyto opravy opět otestovány. Součástí této úrovně jsou jak funkční, tak nefunkční testy.“ [8]

Podle toho, jakým aspektem systému se při testování zabýváme, dělíme testy do kategorií. Na tyto kategorie se dále v textu zaměřím, avšak uvedu zde pouze ty, které se při testování webových aplikací používají nejčastěji.

1.5.3.1 Funkční testování

Funkční testování si klade za cíl nalézt rozdíly mezi chováním vytvořeného systému a předpokládaným chováním, jenž je definováno v dohodnuté specifikaci požadavků (SRS). Jelikož se jedná o testování z vyšší úrovně, používáme k testování typicky techniku černé skříňky.

Podle [6] se zabýváme:

Testování obsahu

Zjišťujeme, zda se na stránce nachází pouze pravdivé informace, a zda jsou u všech tlačítek, obrázků, tabulek správné popisky. Nesmí se stát, že se na stránce nacházejí například špatně kontaktní údaje, nesprávné ceny produktů nebo tlačítka, která dělají něco jiného, než k čemu jsou určena.

Testování odkazů

Při tomto testování ověřujeme, zda jsou všechny odkazy použitelné a zda vedou tam, kam mají. Nemělo by se stát, že na některé stránce nalezneme odkazy, které již nefungují, nebo vedou na nesprávnou stránku.

Testování formulářů

V rámci testování formulářů zjišťujeme, zda pole formulářů obsahují správné výchozí hodnoty a zda přijímají pouze taková data, pro která jsou určena. Po odeslání formuláře se přesvědčíme, že se na stránce zobrazilo správné hlášení, které potvrzuje odeslání formuláře.

Testování databáze

Testování databáze se zabývá kontrolou dotazů, které posíláme do databáze. Cílem je zajistit, že všechny provedené operace měly zamýšlený efekt. Zároveň je nutné zaručit, že po každé práci s databází zůstala data v konzistetním stavu.

1.5.3.2 Testování bezpečnosti

Testování bezpečnosti se soustředí na odolnost systému vůči útokům, které přicházejí z vnějšího prostředí. Mezi nejčastější způsoby narušení bezpečnosti webových aplikací patří například:

Přímý přístup do zabezpečené části

Tento typ útoku předpokládá, že utočník zná adresu do zabezpečené části systému. Chybí-li v systému kontrola pravomocí, může běžný uživatel vcelku snadno provést privilegovanou opraci či náhle získat vyšší oprávnění.

Předání nevalidních parametrů

Útok spočívá v odeslání nevalidních či modifikovaných dat skrze formulář nebo parametry URL adresy. Chybí-li v systému validace vstupních dat, může útočník snadno provést změnu cizích dat, způsobit pád systému, nebo využít útoků SQL injection či Cross-site scripting.

Cross-site request forgery

Tato metoda útoku spočívá v tom, že přimějme uživatele navštívit napadenou webovou stránku, která skrytě provede útok na aplikaci, kde je uživatel přihlášen. Je-li obětí běžný uživatel, může úspěšný útok znamenat například nechtěnou změnu přihlašovacích údajů, aniž by si toho uživatel všiml. Ovšem je-li obětí administrátor systému, může tento útok ohrozit celý systém.

1.5.3.3 Testování výkonu

Úkolem těchto testů je zjistit, jak se vyvíjený systém chová pod různými stupní zátěže. Při testování sledujeme stabilitu a plynulost systému ale také využití prostředků.

Podle [6] do této kategorie patří následující druhy testů:

Zátěžové testování

Zátěžové testování se snaží otestovat chování systému jak při běžném zatížení, tak i při vyšším zatížení, které může simulovat například maximální návštěvnost během špičky. Tento druh testování slouží především k tomu, abychom v systému odhalili slabá místa, která způsobují výkonnostní propady, ale také k tomu, abychom stanovili maximální limity systému.

Stresové testování

Během tohoto testování se snažíme ověřit stabilitu vyvíjeného systému v okamžiku kdy se ocitne pod extrémní zátěží. Testování typicky provádíme v omezeném prostředí, ve kterém se snažíme simulovat podmínky, jenž nevyhovují minimálním parametrům systému. V rámci testů se zabýváme také tím, zda se je systém schopen vyrovnat s pádem některé z jeho částí, respektive zda je schopen tuto část znovu uvést do funkčního stavu.

1.5.3.4 Testování použitelnosti

Testování použitelnosti se zabývá tím, jak se systémem pracují jeho koncoví uživatelé. Podle [9] se během testování zabýváme těmito aspekty uživatelského rozhraní:

- Zvládnutelnost – Jak snadné je pro uživatele s aplikací pracovat?
- Efektivita – Jak dlouho uživateli zabralo, než se s aplikací naučil zacházet?
- Zapamatovatelnost – Jak s aplikací dokáže uživatel pracovat po delší době nepoužívání?
- Spokojenost – Jak příjemné je uživateli aplikaci používat?
- Chybovost – Kolika chyb se během používání aplikace uživatel dopustil?

Z uvedeného je patrné, že tento druh testování se zabývá subjektivním pohledem uživatele na uživatelské rozhraní systému, tudíž jej nelze žádným způsobem automatizovat. Z toho důvodu se tímto druhem testování nebudu více zabývat.

1.5.4 Akceptační testování

Akceptační testování je provedeno zákazníkem při dodání systému. Jeho cílem je ověřit, že dodaný systém splňuje všechny stanované požadavky uvedené ve specifikaci (SRS). Testování je provedeno podle předpřipravených scénářů, na jejichž tvorbě se často podílí také dodavatel vyvíjeného systému. Chyby nalezené během testování jsou hlášeny zpět vývojovému týmu, který v nejkratším možném čase zjedná jejich nápravu. Poté je zákazníkovi dodán nový systém, na kterém je opět provedeno testování. Systému může být nasazen do ostrého provozu až ve chvíli, kdy splní všechna akceptační kritéria stanovená zákazníkem.

1.6 Pravidla pro tvorbu testů

Testy můžeme vytvářet různými způsoby, některé jsou vhodnější, jiné s sebou mohou přinášet spíše problémy. V této podkapitole zmíním určitá pravidla, která nám pomohou tvořit testy mnohem kvalitnější a robustnější.

V [10] je doporučeno pro tvorbu jednotkových testů dodržovat metodiku F.I.R.S.T. Metodika se skládá z několika následujících pravidel:

Rychlý (Fast)

Testy by měly být rychlé. Budou-li pomalé, nebudeme je chtít často spouštět. Tím že je nebudeme spouštět, ztrácejí testy svůj význam. Navíc nenalezneme vzniklé chyby včas, kdy by bylo pro nás jejich řešení výrazně snadnější.

Nezávislý (Independent)

Testy by měly být vzájemně nezávislé. Žádný z testů by neměl spoléhat nejen na podmínky nastavené testy, které byly spuštěny před ním, ale i prostředí, ve kterém jsou testy spouštěny. Z toho plyne, že testy musí být spustitelné v libovolném pořadí, jinak by selhání jednoho testu mohlo způsobit lavinu nefunkčních testů.

Opakovatelný (Repeatable)

Testy musí být opakovatelné. Nelze-li spustit test v libovolném prostředí, které je podobné produkčnímu prostředí, nebo jej můžeme spustit nejvýše jednou, pak je takový test zbytečný, protože jej nebudeme mít možnost spouštět příliš často.

Samovyhodnocující (Self-validating)

Z výstupu testu by mělo být na první pohled zřejmé, jestli prošel anebo selhal. V žádném případě bychom neměli být nuceni číst protokoly a zjišťovat, jak vlastně test dopadl. Z toho plyne, že by stav testu měl být interpretován logickou hodnotou *Ano* či *Ne*.

Aktuální (Timely)

Testy bychom měli vytvořit buď těsně před napsáním zdrojového kódu, nebo ihned po jeho napsání. Pokud budeme testy tvořit později, můžeme zjistit, že je obtížné zdrojový kód testovat, v horším případě, že jej testovat nelze.

Z uvedených pravidel je patrné, že zabývají základními aspekty testů. Proto si myslím, že tato pravidla můžeme aplikovat, alespoň zčásti, i na ostatní druhy testů.

1.7 Shrnutí

Z uvedené analýzy je patrné, že si při testování webových aplikací nevystačíme pouze s jednou testovací technikou. Chceme-li zaručit opravdu kvalitní testování, pak musíme využít jak technik dynamického testování, tak i technik statického testování.

Během testování bychom se měli snažit automatizovat co nejvíce testů. Především kvůli tomu, abychom dříve získali zpětnou vazbu a mohli tak rychleji reagovat na opravu případných chyb. Mimo to se budeme moci zabývat testováním systému také z jiných úhlů pohledu.

Nakonec bychom neměli zapomínat ani na to, že záleží také na způsobu, jakým testy budeme vytvářet. Nebudeme-li se řídit podle určitých pravidel, můžeme za nějaký čas zjistit, že udržování vytvořených testů je pro nás natolik nákladné, že je bude lepší přestat využívat.

Kapitola 2

Systém pro podporu předmětu BI-DBS

V této kapitole se zaměřím na analýzu systému, jímž se tato práce dále zabývá. Nejdříve se seznámíme s předmětem Databázové systémy, pro něž je systém určen. Letmý úvod do problematiky předmětu nám poskytne lepší představu o nárocích kladených na funkcionalitu systému.

V druhé části kapitoly se zaměřím na strukturu systému. Uvedu, z jakých modulů se systém skládá a co tyto moduly řeší. Poté popíši technologie, které byly pro vývoj systému zvoleny, ale také na použitou softwarovou architekturu. Nakonec se zaměřím na prostředí, ve kterém je nasazena produkční verze systému.

Na závěr kapitoly popíši, jakým způsobem bylo řešeno testování systému ještě předtím, než vznikla tato práce.

2.1 Předmět Databázové systémy

V rámci předmětu získají studenti nejen teoretický přehled v oblasti relačních databází, ale i praktické zkušenosti v oblasti návrhu a používání. V průběhu kurzu se naučí navrhovat menší databáze pomocí konceptuálního modelu a následně jej implementovat v databázovém systému. Mimoto se seznámí s architekturou databázových strojů, dotazovacím jazykem SQL a jeho teoretickým protějškem relační algebrou, způsoby uložení dat, transakčním zpracováním a mnohem více.

Pro úspěšné absolvování předmětu musí studenti vytvořit semestrální práci a úspěšně složit dva testy. Cílem semestrální práce je vytvořit databázi menšího informačního systému společně s několika SQL příkazy. Téma semestrální práce není nijak omezené, student si jej volí podle svých představ. Výsledná práce je v průběhu předmětu několikrát hodnocena vyučujícími podle kritérií, která můžeme nalézt v [11].

První test píší studenti v průběhu semestru, zatímco druhý musí splnit v průběhu zkouškového období. Oba testy se skládají z podobných otázek, jejichž cílem je ověřit především praktičtější znalosti studentů. Tedy ty, které studenti získali během tvorby semestrální práce. [11, 12]

2.2 Vznik a vývoj

Vývoj systému započal v roce 2014 v rámci předmětů BI-SP1 a BI-SP2 vyučovaných na Fakultě informačních technologií ČVUT v Praze pod vedením Ing. Jiřího Hunky. Během dvou let se podařilo vyvinout první verzi systému, která byla od letního semestru nasazena do ostrého provozu. Bohužel během ostrého provozu se ukázalo, že systém trpí několika nedostatky, které nebylo snadné odstranit, především kvůli nevhodně zvolené architektuře. Z toho důvodu byl další vývoj této verze systému pozastaven a začalo se uvažovat o tvorbě nástupce, který by se z objevených nedostatků poučil.

Netrvalo dlouho a započal vývoj nového sytému, jímž se zabývali studenti Filip Glazar a Petr Pejša v rámci svých bakalářských prací [1, 2]. Kromě těchto studentů se na vývoji podíleli také studenti z předmětů BI-SP1 a BI-SP2. Společně se jim podařilo během krátké doby vytvořit první funkční verzi nového systému. Jelikož tato verze splňovala očekávání, byla v zimním semestru 2016 uvedena do ostrého provozu[1].

V současné době pokračuje vývoj druhé verze systému. Na jeho vývoji se podílí nové studentské týmy z předmětů BI-SP1 a BI-SP2, ale také studenti, kteří se zabývají určitým aspektem systému v rámci své závěrečné práce.

2.3 Architektura a použité technologie

V této kapitole se zaměřím na použité technologie a zvolenou softwarovou architekturu. Tato analýza bude později využita k tvorbě testovacího prostředí a stanovení vhodných technik pro testování systému.

2.3.1 Použité technologie

Vývoj celého systému je realizován v mnoha programovacích jazycích a technologiích. V této kapitole se omezím pouze na hlavní část systému, jímž je webový portál[1]. Na zbylé části systému se zaměříme v kapitole 2.3.3.

Webový portál je vytvořen v programovacím jazyku PHP, respektive je postaven na webovém frameworku Nette. Pro uložení trvalých dat je používán relační databázový systém PostgreSQL.

Nette framework je populární český framework umožňující tvorbu webových aplikací v PHP. Zaměřuje se především na čistý objektový návrh, dokonalé zabezpečení a rychlost. Framework je rozdělen do komponent, které lze

[1]Dostupné na adrese `https://dbs.fit.cvut.cz`

používat nezávisle na sobě, což zvyšuje flexibilitu celého frameworku. Mezi tyto komponenty patří například šablonovací systém Latte, nástroj na debugování a zpracování chyb Tracy, knihovna pro práci s daty ve formátu NEON a mnoho dalších.

Zpočátku této práce bylo pro vývoj webového portálu používáno starší PHP, konkrétně verze 5.6. V průběhu práce však došlo k aktualizaci jazyka na verzi 7.1. Tato verze s sebou přinesla mnoho změn nejen v rychlosti a syntaxi, ale i v bezpečnosti jazyka. Jedním z hlavních přínosů této verze je bez pochyb podpora silného typování. Zavedením datových typů lze z aplikace, do určité míry, odstranit nepředvídatelné chování jazyka, což nám umožní tvořit bezpečnější a stabilnější aplikace.

2.3.2 Softwarová architektura

Webový portál využívá vícevrstvou architekturu, konkrétně návrhový vzor Model-View-Controller (MVC).

Základní myšlenkou návrhového vzoru MVC je oddělit aplikační logiku od výstupu. Celá aplikace je tedy rozdělena do tří vrstev: řídící logiku (controller), datový model (model) a uživatelské rozhraní (view). Díky tomuto rozdělení získáme lépe strukturovaný zdrojový kód, který je navíc lépe testovatelný a udržovatelný.

2.3.2.1 Model

Model je vrstva, ve které je umístěna hlavní logika aplikace. To znamená, že právě model je zodpovědný za komunikací s externími systémy, výpočetní operace či změny stavů různých objektů. Tato vrstva není vázána na žádnou z ostatních vrstev. Z toho plyne, že pokud bychom chtěli tuto vrstvu použít v jiné aplikaci, měli bychom být schopni ji ze systému snadno vyjmout a následně ji bez větších obtíží použít.

2.3.2.2 View

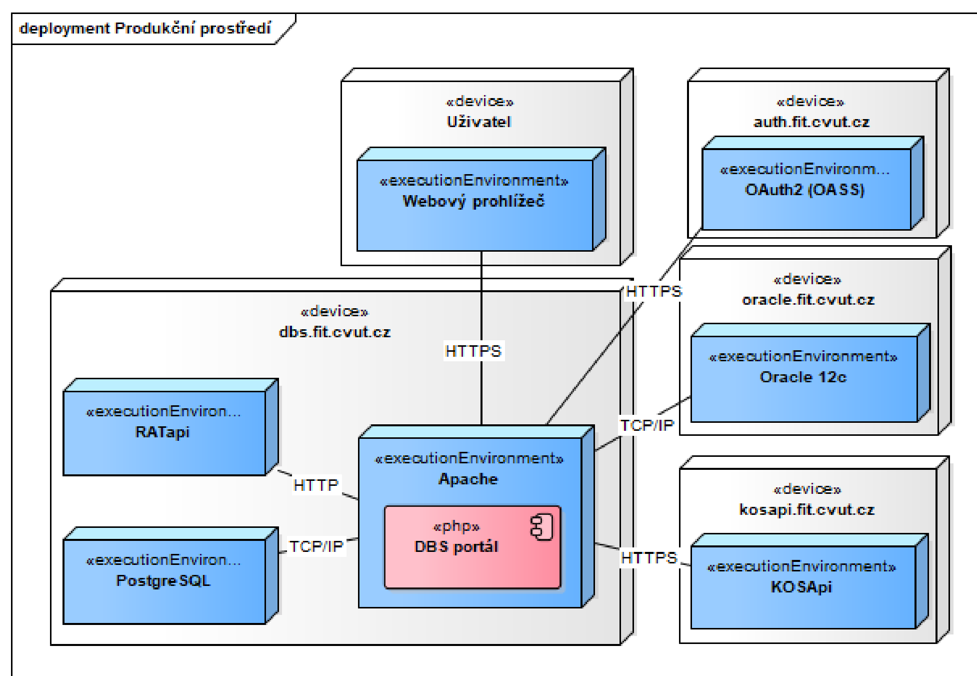
Tato vrstva má na starosti uživatelské rozhraní a zobrazení dat koncovému uživateli. V kontextu webových aplikací je tvořena především kombinací jazyků HTML, CSS a JavaScript. V pokročilých frameworcích může být tvořena skripty pro šablonovací systém, které jsou však následně přeloženy do dříve zmíněných jazyků.

2.3.2.3 Controller

Kontrolér zodpovídá za komunikaci mezi modelovou vrstvou a pohledem (view), ale také za interakci s uživatelem. Požadavky přicházející od uživatelů deleguje na příslušnou obsluhu v modelové vrstvě. Jakmile dojde v modelové vrstvě ke zpracování požadavku, požádá view o vykreslení příslušné odpovědi.

2.3.3 Produkční prostředí

Jak již bylo zmíněno dříve, celý systém se skládá z více komponent, které jsou dále vázány na služby třetích stran. Na obrázku 2.1 je uveden diagram nasazení systému včetně vazeb na externí systémy.



Obrázek 2.1: Diagram nasazení systému v produkčním prostředí včetně vazeb na externí systémy

2.3.3.1 Externí systémy

Z uvedeného obrázku 2.1 je patrné, že systém potřebuje ke svému běhu následující externí rozhraní: OAuth2 [13], KOSApi [14] a databázový server Oracle.

Systém OAuth2 je autorizační server, jenž slouží pro získání autorizačních klíčů, díky nímž lze komunikovat s jinými fakultními systémy. V tomto systému se využívá především pro vytvoření komunikace s rozhraním KOSApi.

Závislost na rozhraní KOSApi je neméně důležitá. Díky vazbě na toto API můžeme do systému importovat nová data o uživatelích, kurzech, paralelkách či zkouškách.

Databázový systém Oracle je v rámci systému využíván při aktivitách týkajících se semestrálních prací a testů. Podrobnější využití tohoto systému bude uvedeno v kapitole 2.4.

2.3.3.2 Portál dbs.fit.cvut.cz

V době psaní této práce byl webový portál nasazen na školním serveru, jehož parametry jsou uvedeny v tabulce 2.1.

CPU	Intel(R) Xeon(R) E5-2630 @ 2.30 GHz
RAM	8 GB
HDD	50 GB
OS	Debian Jessie 8.4 64bit

Tabulka 2.1: Parametry serveru

Z obrázku 2.1 si můžeme povšimnout, že mimo webového serveru Apache je na tomto serveru přítomna také komponenta RATapi, jejímž autorem je Martin Kubiš [15]. Úkolem této komponenty je překládat dotazy z relační algebry do jazyka SQL.

2.4 Funkcionalita systému

V této kapitole se zaměřím na popis funkcí, jimiž webový portál disponuje. Vzhledem k tomu, že cílem této práce není provést detailní analýzu portálu, zaměřím se pouze na stručný popis modulů, z nichž se portál skládá. Předtím se však podíváme na funkcionalitu, která je pro všechny moduly společná.

2.4.1 Společný základ

Mezi společné funkcionality systému patří například správa a přihlášení uživatelů, obsluha notifikací či volba jazykové mutace portálu. Tyto funkcionality jsou jistě nezbytné pro chod webového portálu, avšak z hlediska ostatních modulů jsou spíše nepotřebné. Hlavní funkcionalitou, která je využívána ostatními moduly, je rozhraní, jenž poskytuje přístup k datům uživatelů, paralelek a kurzů.

2.4.2 Administrace

Do administrace systému mají přístup pouze uživatelé, kteří disponují rolí *administrátor*. V rámci administrace mohou uživatelé skrz KOSApi importovat do systému nové kurzy, spravovat uživatele a uživatelské role či se přihlásit za jakéhokoliv uživatele v systému.

2.4.3 Správa databází

Tento modul umožňuje uživatelům vytvářet, editovat či mazat připojení do externích databází. Takto vytvořená databázová spojení jsou poté sdílena v rámci celého webového portálu. Uživatelé je tedy mohou využít například

k vykonání svých databázových dotazů či skriptů. V době psaní této práce bylo možné vytvořit spojení pouze s databázovými systémy Oracle.

2.4.4 Semestrální práce

Tento modul si klade za cíl umožnit studentům vytvářet a odevzdávat jejich semestrální práce a zároveň umožnit učitelům opravu odevzdaných prací.

Samotná semestrální práce se skládá za několika částí, jako je volba tématu, skripty pro naplnění databáze, ukázkové SQL dotazy, aj. Všechny tyto části jsou hodnoceny podle kritérií, která může uživatel s rolí *garant* změnit v nastavení modulu.

Jakmile student dokončí svojí semestrální práci, může ji nechat zkontrolovat automatem. Automat práci vyhodnotí podle stanovených pravidel. Po dokončení opravy si může student práci zobrazit a zjistit, ve kterých částech se dopustil chyb. Není-li student s hodnocením spokojen, může semestrální práci opravit a nechat znovu zkontrolovat. V opačném případě ji může poslat svému učiteli k hodnocení.

Učitelé mohou odevzdané práce buď ohodnotit, nebo odmítnout. Semestrální práci mohou odmítnout pouze v případě, že nejde o řádné odevzdání.

2.4.5 Testy

Modul testů se zabývá tvorbou, psaním a hodnocením zápočtových a zkouškových testů.

Učitelé mohou v systému vytvářet testové otázky různých typů. Takto vytvořené otázky poté mohou použít v testových šablonách, které symbolizují nespuštěný test. Z testové šablony mohou následně spustit test studentům, které si zvolí.

Jakmile učitel spustí test, studentům, kteří do spuštěného testu patří, se uzamkne portál tak, aby mohli pracovat pouze s modulem testů. V okamžiku kdy studenti test vyplní a odevzdají, portál se znovu odemkne. Mimo jiné se při odevzdání testu provede také automatikcká oprava.

Automatická oprava spočívá ve vyhledání shodných odpovědí, které jsou již ohodnoceny. Je-li v systému nalezena shodná odpověď, ohodnotí se studentova odpověď shodným počtem bodů. Není-li v systému nalezena shodná odpověď, předá se studentova odpověď učiteli k manuálnímu hodnocení. Jakmile učitel tuto odpověď ohodnotí, vyhledají se v systému podobné neohodnocené odpovědi a ohodnotí se shodným způsobem.

2.5 Stav testování

Před zahájením této práce byl webový portál testován především manuálně. K dispozici bylo také několik automatizovaných jednotkových a akceptačních testů. Tyto testy však byly závislé na okolním prostředí, a proto je nebylo po určité době možné znovu spustit.

K tvorbě jednotkových testů byl používán nástroj Nette/Tester [16]. Výhodou tohoto nástroje je dobrá integrace s použitým Nette frameworkem, avšak bylo jej možné využít pouze k tvorbě jednotkových testů. Před zahájením této práce bylo pro testování portálu připraveno celkem třicet jednotkových testů. Naneštěstí část z nich již nebylo možné spustit.

Akceptační testy byly vytvořené v nástroji Selenium IDE [17]. Připraveno bylo celkem jedenáct testovacích scénářů, které však nebylo možné v době psaní této práce automaticky spustit, právě díky jejich závislostem na okolním prostředí.

Samotné testování probíhalo v lokálních prostředích vývojářů. Naneštěstí každý z vývojářů měl své vlastní vývojové prostředí, tudíž nebylo možné zaručit, že otestovaný kód bude správně pracovat i v produkčním prostředí.

Kapitola 3

Návrh

V této kapitole se budu zabývat návrhem vhodných postupů pro testování webového portálu, jenž byl představen v minulé kapitole. Nejprve se zaměřím na návrh testovacího prostředí. Popíši, jak bude toto prostředí vytvořeno a naznačím jaké bude jeho využití.

Dále se zaměřím na volbu testovacích technik. Popíši, jaké techniky budou zvoleny z oblasti statického testování a jaké z oblasti dynamického testování.

V závěru kapitoly se zaměřím na průběžné testování a automatické nasazení. Popíši, jakým způsobem bude průběžné testování řešeno a jak jej bude možné využít při automatickém nasazení portálu do produkčního prostředí.

3.1 Testovací prostředí

Jak již bylo řečeno v kapitole 1.2.3, chceme-li zaručit spolehlivé testování vyvíjeného systému, musí být vytvořené testovací prostředí shodné s produkčním prostředím. Pokud by se tato prostředí výrazněji lišila, může čas od času docházet k tomu, že systém bude v jednom prostředí pracovat dle očekávání, zatímco v druhém prostředí bude vykazovat chyby. Naneštěstí vytvořit dvě naprosto shodná prostředí není snadné. Můžeme však využít virtualizačních technologií, které nám s tvorbou prostředí pomohou.

V této práci budou vytvořena celkem dvě prostředí. Jedno z prostředí bude určeno pro vývojáře projektu, tudíž bude navíc obsahovat nástroje, které jsou nezbytné pro vývoj webového portálu. Z toho je patrné, že toto prostředí nebude testovací, nýbrž vývojové. Druhé prostředí bude využíváno při průběžném testování webového portálu. Toto prostředí bude shodné s produkčním prostředím, avšak navíc bude obsahovat nástroje, které jsou nutné pro testování vyvíjeného portálu.

Vzhledem k tomu, že tato prostředí potřebujeme distribuovat mezi ostatními vývojáři projektu, a zároveň chceme, aby je bylo možné vytvořit bez větších obtíží v jakémkoliv prostředí, budou k jejich vytvoření využity virtualizační technologie Vagrant [18] a Docker [19].

3.2 Testovací techniky

Vzhledem k rozsahu webového portálu a použitým technologiím jsem se rozhodl, že bude testování zahrnovat jak techniky dynamického testování, tak i techniky statického testování.

Pro využití technik statického testování jsem se rozhodl proto, že nás dokáží upozornit na případné chyby ihned pro napsání zdrojového kódu, tedy ještě předtím, než vytvoříme jakýkoliv testovací případ. Tento druh testování však dokáže odhalit pouze určité druhy chyb, a proto jsem se rozhodl zvolit také techniky dynamického testování. Díky zařazení technik dynamického testování dokážeme nalézt rozdíly mezi chováním vytvořené funkcionality a jejím popisem ve specifikaci požadavků (SRS).

3.2.1 Statické testování

Z oblasti statického testování budou využity pouze techniky, které umožňují automatizované testování zdrojového kódu. Proto se bude testování zabývat pouze formou a obsahem vytvořeného kódu.

3.2.1.1 Kontrola formy

Před vznikem této práce nebyla v projektu stanovena žádná pravidla, která by vývojářům říkala, jak mají formátovat či strukturovat jejich zdrojový kód. Každý z vývojářů se tedy řídil podle svých vlastních pravidel. Naneštěstí díky tomuto přístupu vznikal sice funkční, ale nepříliš čitelný zdrojový kód, jenž bylo často obtížné pochopit.

Z toho důvodu jsem se rozhodl, že součástí této práce bude také vytvoření kódovacího standardu, v němž bude popsáno, jak správně formátovat a strukturovat zdrojový kód. Aby se tento kódovací standard příliš nelišil od jiných projektů vytvořených v jazyce PHP, budu při jeho návrhu vycházet ze zažitých konvencí jazyka PHP.

3.2.1.2 Kontrola obsahu

Kontrola obsahu se bude týkat pouze souborů, které jsou ve formátu PHP, Latte a NEON. Tedy souborů, které jsou spjaté s Nette frameworkem. Samotné testování bude rozděleno do dvou fází.

První fáze kontroly bude zaměřena na ověření syntaxe všech souborů v uvedených formátech. Pro kontrolu syntaxe budou využity výhradně oficiální knihovny, které umožňují intepretaci těchto jazyků.

Druhá fáze kontroly se bude týkat pouze souborů v jazyce PHP. Tato kontrola se bude zabývat nalezením chyb, které jsou způsobené například používáním nedefinovaných proměnných či předáváním nevhodných argumentů do funkcí a metod. Analýza zdrojových souborů bude zajištěna nástrojem PHPStan, jehož plnou funkcionalitu popíši později.

3.2.2 Dynamické testování

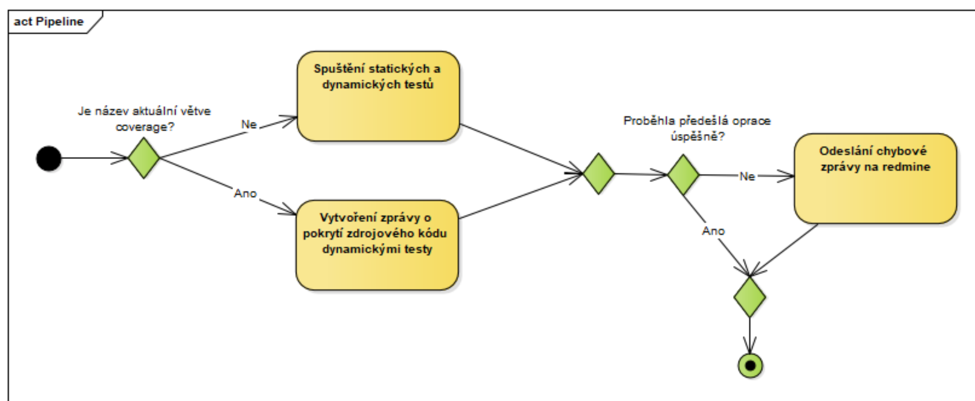
Před vznikem této práce bylo možné testovat webový portál pomocí několika jednotkových a akceptačních testů. Tyto testy využívaly ke svému běhu nástroje Nette/Tester a Selenium IDE.

Podle mého názoru nelze pomocí zvolených technik spolehlivě otestovat celý portál. Proto jsem se rozhodl, že testování bude rozšířeno o bezpečnostní, funkční a integrační testování. Vzhledem k tomu, že se bezpečnostní a funkční testování zčásti překrývají, bude testování bezpečnosti zahrnuto do funkčních testů.

Jelikož pro nově zvolené testovací techniky nelze využít aktuálně používané nástroje, rozhodl jsem se, že budou nahrazeny frameworkem Codeception [20]. Naneštěstí tento framework nedisponuje plnou podporou Nette frameworku, a proto jsem se rozhodl, že v rámci této práce vytvořím také modul, který umožní oba zmíněné frameworky propojit.

3.3 Průběžné testování

Zdrojový kód webového portálu bude automaticky testován při každém vytvoření nové revize ve společném repozitáři projektu. Tato frekvence testování byla zvolena proto, aby bylo možné odhalit případné chyby ihned po jejich propagaci do společného repozitáře.



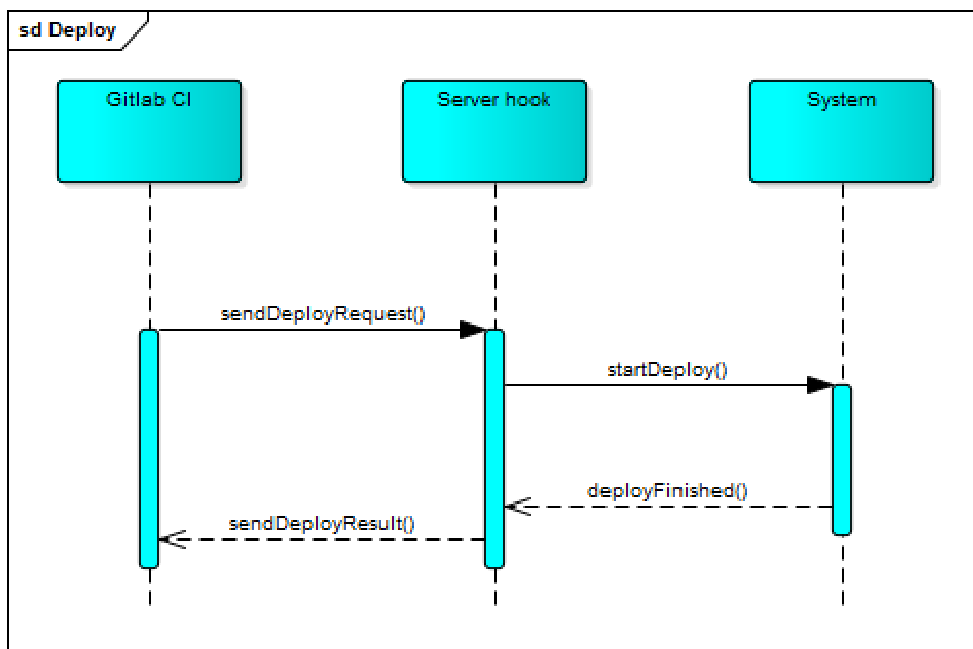
Obrázek 3.1: Předpokládaný průběh testování při vytvoření nové revize v repozitáři projektu

Na obrázku 3.1 je znázorněn předpokládaný průběh celého testování. Implementace uvedeného diagramu bude zajištěna nástrojem Gitlab CI [21]. Tento nástroj byl zvolen jednak kvůli dobré integraci s nástrojem Gitlab, který je v rámci tohoto projektu používán pro verzování zdrojového kódu, ale také kvůli tomu, že umožňuje spustit testování ve virtualizovaném prostředí.

3.4 Automatické nasazení

Již před započetím této práce bylo, do určité míry, nasazení nového aplikačního kódu automatizováno. O nasazení kódu se starala sada skriptů, která se automaticky spustila po stažení aktualizace v produkčním prostředí. Aktualizace se však musela spustit manuálně.

Cílem této práce bude vytvořit rozhraní, které umožní spustit proces nasazení v produkčním prostředí ihned po skončení testování. V rámci práce bude vytvořen jednak klient, který odešle žádost o nasazení do produkčního prostředí, ale také server, který tento požadavek dokáže zpracovat a na jeho základě spustit vhodné skripty pro nasazení portálu. Pro lepší představu je předpokládaný proces nasazení zachycen na obrázku 3.2.



Obrázek 3.2: Předpokládaný proces nasazení

Kapitola 4

Realizace

V první části této kapitoly se podrobněji zaměřím na implementaci návrhu, jenž byl představen v předchozí kapitole. V závěru kapitoly zhodnotím výsledky provedeného testování.

4.1 Tvorba prostředí

Na základě návrhu, jenž byl představen v předchozí kapitole, byla vytvořena dvě virtualizovaná prostředí – vývojové a testovací. Vzhledem k tomu že tato prostředí necílí na shodnou skupinu uživatelů, bylo každé z prostředí vytvořeno v jiném virtualizačním nástroji.

4.1.1 Požadavky na prostředí

Abychom zmíněná prostředí mohli využít pro testování systému, musí splňovat minimální požadavky jak pro běh webového portálu (viz. kapitola 2.3.3), tak i pro běh samotných testů. Z toho důvodu jsou v obou prostředích nainstalovány následující nástroje:

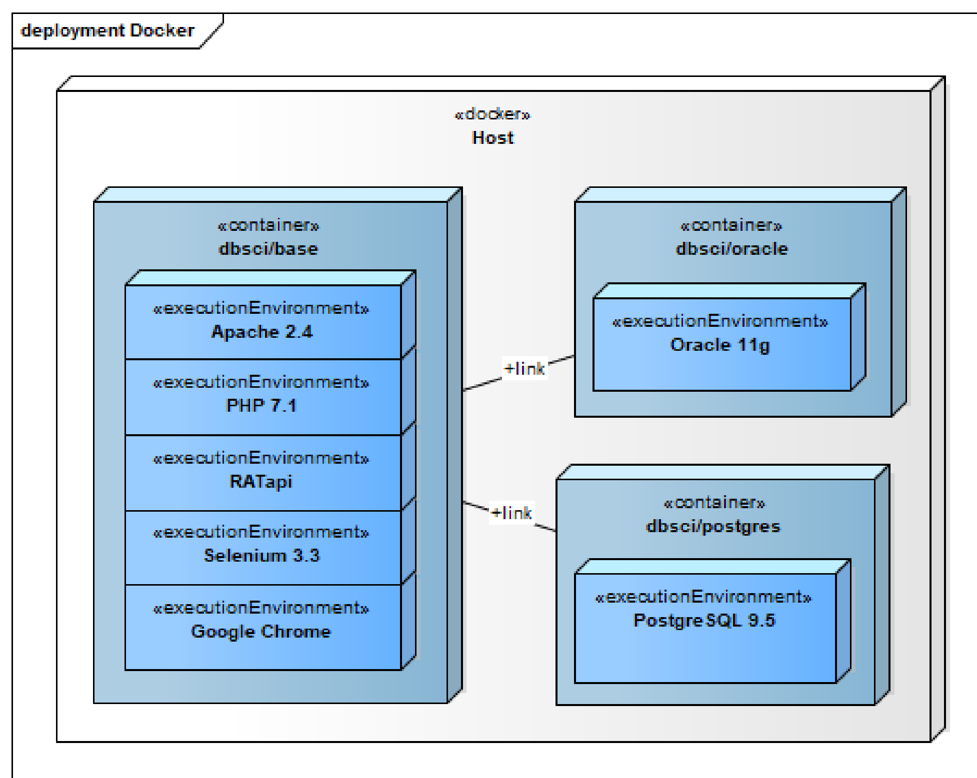
- Webový server Apache 2.4
- PHP 7.1 s rozšířeními PDO_PGSQL, PDO_OCI, OCI8 a Xdebug
- Databázové systémy PostgreSQL 9.5 a Oracle 11g
- Překladač relační algebry (RATapi)
- Selenium server 3.3
- Webový prohlížeč Google Chrome 58 s ovladačem chromedriver 2.29

Po instalaci je nutné tyto nástroje vhodně nakonfigurovat. Předpokládaná konfigurace je popsána v dokumentaci na přiloženém CD.

4.1.2 Testovací prostředí

Pro tvorbu testovacího prostředí byl zvolen virtualizační nástroj Docker kvůli tomu, že poskytuje dobrou integraci s nástrojem Gitlab CI, jenž byl využit pro implementaci procesu průběžného testování.

Celé testovací prostředí je sestaveno ze tří samostatných obrazů, které jsou mezi sebou propojené. Pro lepší představu čtenáře je složení prostředí znázorněno na obrázku 4.1.



Obrázek 4.1: Vizualizace testovacího prostředí vytvořeného v nástroji Docker

Vzhledem k tomu, že uvedené obrazy je nutné čas od času aktualizovat, není příliš vhodné je vytvářet manuálně. Z toho důvodu bylo v rámci této práce vytvořeno několik skriptů, které celý proces sestavení obrazů automatizují. Zmíněné skripty a materiály popisující jejich použití jsou umístěny na přiloženém CD.

4.1.3 Vývojové prostředí

Vývojové prostředí bylo vytvořeno pomocí nástroje Vagrant. Tento nástroj byl zvolen kvůli tomu, že na rozdíl od nástroje Docker poskytuje lepší integraci s ostatními nástroji využívanými při vývoji systému.

Celé prostředí je složeno ze dvou virtuálních strojů, které je možné využívat nezávisle na sobě. V hlavním virtuálním stroji jsou dostupné všechny nástroje nutné pro vývoj a běh webového portálu, zatímco v druhém virtuálním stroji je umístěn pouze databázový systém Oracle. Zvídavý čtenář se jistě zeptá, proč bylo zvoleno toto rozdělení. Důvod je prostý, databázový systém Oracle má vysoké nároky na prostředky operačního systému, a proto není vhodné, aby běžel nepřetržitě, i když jej zrovna nevyužíváme.

I tyto virtuální stroje je nutné čas od času aktualizovat. Proto byla, podobně jako u testovacího prostředí, vytvořena sada skriptů, které sestavení virtuálních strojů automatizují. Vytvořené skripty jsou opět včetně podpůrných materiálů umístěny na přiloženém CD.

4.2 Vývoj podpůrných nástrojů a modulů

V této podkapitole se zaměřím na popis nástrojů vytvořených v rámci této práce. Zabývat se budu jednak nástroji představenými v návrhu, ale také nástroji, které vznikly až podle skutečných potřeb v průběhu práce.

4.2.1 Statické testování

Jak již bylo řečeno v návrhu, statické testování je zaměřeno pouze na kontrolu formy a obsahu zdrojového kódu. O kontrolu formy se stará kódovací standard, který popíši později.

Kontrola obsahu byla zpočátku zajištěna nástrojem nette/code-checker [22]. V průběhu práce se však ukázalo, že některé kontroly, provedené tímto nástrojem, jsou v rozporu se stanoveným kódovacím standardem, a proto se od jeho používání upustilo. Avšak některá validační pravidla byla natolik užitečná, že jsem se je rozhodl implementovat jako samostatné moduly. Konkrétně se jedná o validační pravidla pro jazyka Latte a NEON.

4.2.1.1 Kodovací standard

Během návrhu kódovacího standardu byl kladen především důraz na dodržení zažitých konvencí jazyka PHP. Z toho důvodu byl jako podklad zvolen kódovací standard PSR-2 [23], který byl rozšířen o další pravidla.

V průběhu práce však došlo k aktualizaci jazyka PHP na verzi 7.1 a vytvořený kódovací standard přestal být dostačující a to proto, že nepokrýval poslední syntaxi jazyka. Proto jsem se rozhodl rozšířit již používaný kódovací standard o další pravidla z balíčku slevomat/coding-standard [24]. Díky tomu se mi podařilo vytvořit plnohodnotný kódovací standard pro jazyk PHP 7.1.

Výsledná podoba kódovacího standardu je umístěna jednak v příloze B, ale také v dokumentaci na přiloženém CD.

4.2.1.2 Validátor jazyka NEON

Úkolem tohoto validátoru je ověřit, že předložené soubory splňují syntaxi jazyka NEON. Kvůli spolehlivosti je pro ověření syntaxe používán oficiální analyzátor z balíčku nette/neon [25]. Z toho je patrné, že v rámci této práce byla vytvořena pouze vrstva, která umožní spustit analyzátor z příkazové řádky.

4.2.1.3 Validátor jazyka Latte

Cílem validátoru je jednak ověřit, že předložené soubory splňují syntaxi šablonovacího jazyka Latte ale také, že používají pouze existující makra. Samotná validace je zajištěna oficiálním kompilátorem z balíčku latte/latte [26]. Z uvedeného je zřejmé, že v rámci této práce byla vytvořena pouze vrstva, která umožní spustit kompilátor z příkazové řádky.

4.2.2 Dynamické testování

Jak již bylo uvedeno v návrhu, pro tvorbu dynamických testů byl zvolen testovací framework Codeception. Během testování se však ukázalo, že zvolený framework neposkytuje dostatečnou funkcionalitu pro testování celého sytému, a proto bylo vytvořeno několik podpůrných modulů, které chybějící funkcionalitu doplní.

4.2.2.1 Modul Nette

Cílem tohoto modulu je umožnit funkční testování webových aplikací vytvořených v Nette frameworku. Během testování simuluje modul chování webového serveru tím, že při zpracování požadavku na změnu stránky provede znovu spuštění celé Nette aplikace. Modul zároveň zajišťuje izolaci databáze tím, že veškeré operace poslané do databáze jsou vykonány v databázové transakci.

Vývoj celého modulu byl rozdělen do dvou fází. V první fázi vývoje byl vytvořen prototyp modulu. Během vývoje byl kladen důraz především na implementaci funkcí bez ohledu na efektivnost. V průběhu testování se ukázalo, že vytvořený modul splňuje očekávání, avšak při vyšším počtu testů rychle rostou nároky na prostředky operačního systému.

Druhá fáze vývoje modulu byla zaměřena především na optimalizaci a eliminací úniků paměti. Díky provedeným optimalizacím se podařilo při 1 400 testech snížit paměťové nároky téměř o 80 % a délka testování se zkrátila přibližně o 30 %.

Celková funkcionalita poskytovaná tímto modulem je popsána v dokumentaci na přiloženém CD.

4.2.2.2 Modul Faker

Cílem tohoto modulu je umožnit generování náhodných dat v rámci dynamických testů. Modul je implementován jako rozhraní mezi testovacím frameworkem a knihovnou fzaninotto/Faker [27].

4.2.2.3 Modul Migrations

Úkolem tohoto modulu je umožnit obnovu testovací databáze před spuštěním jednoho či více dynamických testů. Modul je implementován jako vrstva nad knihovnou nextras/migrations [28], jenž je v rámci projektu používána pro správu produkční databáze.

4.2.2.4 Modul MultiDb

Tento modul nabízí podobnou funkcionalitu jako oficiální modul Db [29]. Hlavní rozdíl je však v tom, že tento modul dokáže navázat spojení s více databázemi současně. Díky tomu je možné provést validaci uložených dat i v externích databázích.

4.3 Průběžné testování

Webový portál je průběžně testován při každém nahrání nového zdrojového kódu do společného repozitáře projektu. Pro implementaci celého procesu byl zvolen nástroj Gitlab CI, který pro spuštění testů využívá virtualizované testovací prostředí vytvořené v nástroji Docker. Díky tomu jsou všechna testování vzájemně nezávislá.

Celé testování lze rozdělit do dvou hlavních fází. V první fázi testování jsou spuštěny testy z oblasti statického testování, zatímco v druhé fázi testování jsou spuštěny všechny testy z oblasti dynamického testování. V okamžiku, kdy některý z testů selže, je celé testování předčasně ukončeno. Současně s tím je odesláno upozornění do nástrojů Slack a Redmine. Tyto nástroje jsou v rámci projektu používány pro komunikaci vývojářů a správu úkolů, díky tomu je vyšší pravděpodobnost, že si autor chyby neúspěšného testování všimne.

Výslednou konfiguraci nástroje Gitlab CI lze nalézt v souboru *.gitlab-ci.yml* na přiloženém CD ve složce se zdrojovými kódy projektu.

Pro lepší představu čtenáře je celý průběh testování zachycen na obrázku C.1, jehož části postupně popíši.

4.3.1 Kontrola syntaxe PHP souborů

V první fázi průběžného testování dochází k ověření syntaxe všech PHP souborů. K ověření syntaxe je využíván nástroj Jakub-Onderka/PHP-Parallel-Lint [30], který validuje soubory paralelně, čímž značně spoří celkový čas testo-

vání. Ve skutečnosti tento nástroj používá k validaci oficiální PHP interpreter, díky čemuž je zaručeno dokonalé pokrytí PHP syntaxe.

4.3.2 Kontrola syntaxe NEON a Latte souborů

O kontrolu syntaxe souborů ve formátu NEON a Latte se starají nástroje, které vznikly v rámci této práce. Chování těchto nástrojů je popsáno v kapitolách 4.2.1.2 a 4.2.1.3.

4.3.3 Kontrola formátování PHP kódu

V rámci této fáze průběžného testování dochází k ověření formy zdrojového kódu dle stanoveného kódovacího standardu. O ověření stanovených pravidel se stará nástroj squizlabs/PHP_CodeSniffer [31].

4.3.4 Statické analýza nástrojem PHPStan

Poslední fáze statického testování je zaměřena na hledání chybných konstrukcí ve zdrojových souborech jazyka PHP. Již z názvu podkapitoly je zřejmé, že se o testování stará nástroj PHPStan. Cílem tohoto nástroje je nalézt ve zdrojovém kódu následující druhy chyb:

- Existence tříd a rozhraní v konstrukcích instanceof, catch, anotacích či datových typech.
- Existence proměnných v rámci cyklů a podmínek.
- Existence a viditelnost volaných metod a funkcí.
- Existence a viditelnost používaných proměnných a konstant.
- Správné přiřazení do proměnných dle datových typů.
- Správné předání parametrů konstruktorům, metodám a funkcím.
- Správné návratové hodnoty z metod a funkcí.
- Nepotřebné přetypování – například (string) 'foo'.
- Nepotřebné parametry v konstruktoru, které nejsou dále využity.

[32]

4.3.5 Dynamické testování nástrojem Codeception

V poslední fázi průběžného testování jsou spuštěny všechny dynamické testy. Abychom zajistili, že jsou testy vykonávány od nejnižší úrovně abstrakce, spouštíme je v následující pořadí: jednotkové testy, integrační testy, funkční testy, a nakonec akceptační testy.

4.3.6 Odeslání stavu testování na Slack

V okamžiku, kdy některá z částí testování skončí s neúspěchem, dochází k automatickému odeslání upozornění do komunikačního kanálu #ci_build v nástroji Slack. Díky tomu jsou vývojáři včas informováni o možném výskytu chyb a mohou zjednat jejich nápravu.

O odeslání zmíněného upozornění se stará rozšíření v nástroji Gitlab. Z toho je zřejmé, že v rámci této práce byla provedena pouze vhodná konfigurace zmíněného rozšíření.

4.3.7 Zapsání stavu testování do Redmine

Skončí-li některá z částí testování neúspěchem a v popisu revize (v rámci verzovacího nástroje Gitlab) je uveden identifikátor úkolu ve formátu [#číslo], pak je toto testování zaznamenáno také k příslušnému úkolu do nástroje Redmine. Zápis je proveden formou komentáře od uživatele Gitlab bot. Součástí komentáře je identifikátor chybné revize, jméno ovlivněné větve v rámci verzovacího systému a autor chyby.

O zapsání zmíněného komentáře se stará skript, který vznikl v rámci této práce. Úkolem tohoto skriptu je přečíst nastavené globální proměnné prostředí a na jejich základě poslat požadavek o vytvoření nového komentáře na rozhraní nástroje Redmine. Vytvořený skript je umístěn na přiloženém CD ve složce s ostatními skripty pro sestavení testovacího prostředí.

4.4 Automatické nasazení

Pro účely automatického nasazení byla vytvořena klient-server aplikace, jejímž cílem je spustit v produkčním prostředí skripty, které zajistí nasazení nového aplikačního kódu.

Serverová část přestavuje jednoduché REST API, které přijímá POST požadavky na adrese /. Při zpracování požadavku je ověřeno, že jej klient zaslal s platným autentizačním klíčem. Proběhne-li ověření úspěšně, pak je v produkčním prostředí spuštěn skript, který nové změny automaticky nasadí. Poté je klientovi odeslána zpráva, ve které je popsán obsahuje stav nasazení – úspěch či neúspěch. Ovšem je-li ověření autentizačního klíče neúspěšné, pak je klientovi ihned odeslána zpráva, ve které popsáno, že ověření klíče selhalo.

Klientská část je realizována jako jednoduchý skript, který přečte hodnoty nastavených globálních proměnných a na jejich základě odešle požadavek na nasazení nové verze projektu. Před odesláním požadavku je však zkontrolováno, že se snažíme nasadit pouze změny, které se nacházejí ve větvích (verzovacho systému) souvisejících s produkčním prostředím.

V praxi však tento přístup nebyl nikdy použit kvůli tomu, že produkční prostředí sdílí databázi s testovacím prostředím[2]. Jakákoliv nevhodná změna

[2]Dostupné na: https://dbs2.fit.cvut.cz

ve struktuře databáze může způsobit nefunkčnost jednoho z prostředí, a proto je nasazení nového aplikačního kódu spouštěno pouze manuálně.

Přesto je však vytvořená klient-server aplikace včetně dokumentace umístěna na přiloženém CD.

4.5 Zhodnocení testování

Webový portál byl testován jak manuálně, tak i pomocí automatizovaných testů. Manuální testování bylo, do určité míry, využíváno při návrhu testovacích scénářů pro funkční a akceptační testy.

Celkem bylo v rámci práce vytvořeno 1 415 automatizovaných testů, které pokrývají přibližně 70 % zdrojového kódu. Vyššího pokrytí automatizovanými testy by nebylo snadné dosáhnout, protože webový portál je stále ve fázi aktivního vývoje, tudíž některé funkcionality byly během testování buď zcela přepracovány, nebo byly přidány až po dokončení testování. Přesto však, podle mého názoru, dokáží vytvořené testy spolehlivě otestovat základní funkcionalitu webového portálu. Detailní pokrytí zdrojového kódu automatickými testy lze nalézt v dokumentaci na přiloženém CD.

Během testování byly objeveny jednak chyby ve funkcích webového portálu, ale také v knihovnách, které testovaný portál využívá ke svému běhu. Celkový počet nalezených chyb je shrnut v tabulce 4.1.

Projekt	**Chyb**	**Návrhů na zlepšení**
Webový portál dbs.fit.cvut.cz	91	97
Framework Codeception	2	-
Nástroj PHPStan	1	-
Jazyk PHP	1	-
Knihovna nette/http	1	-
Knihovna fzaninotto/Faker	1	-
Knihovna mcustiel/Creature	1	-
Celkem	98	97

Tabulka 4.1: Statistika nahlášených chyb a návrhů na zlepšení

V rámci testování byly odhaleny jednak chyby, které způsobovaly neočekávané či nezamýšlené chování webového portálu, ale také nedostatky na úrovni zdrojového kódu. Objeveny byly především následující druhy chyb:

Vysoká provázanost

Během jednotkového testování se ukázalo, že je tento druh testů velmi obtížné aplikovat na již vytvořený zdrojový kód. Vytvořené třídy obsahují mnoho skrytých závislostí, které při jejich testování není možné žádným způsobem nahradit za vlastní implementací. Z toho důvodu byla jednotkovými testy otestována pouze malá část webového portálu.

Nedodržení architektury

Při nízkoúrovňovém testování (tj. při jednotkovém a integračním testování) bylo v mnoha částech zdrojového kódu zjištěno, že vývojáři příliš nedodržují zvolenou MVC architekturu. Nejčastěji bývá sloučena řídící a modelová vrstva. Důsledkem tohoto sloučení je jednak horší udržovatelnost zdrojového kódu, ale také horší testovatelnost.

Neošetřená vstupní data

Během testování bylo zjištěno, že webový portál často nekontroluje obdržená vstupní data. Díky tomu bylo možné, například změnou formulářových polí či změnou parametrů URL adresy, v systému vyvolat neočekávané chování nebo pád.

Nedostatečné zabezpečení

Při bezpečnostním testování bylo odhaleno, že webový portál příliš nekontroluje pravomoce uživatelů. To znamená, že pokud koncový uživatel zná adresu směřující do zabezpečené části, nebo dokáže odhadnout identifikátory různých zdrojů, pak může snadno získat přístup k datům cizího uživatele nebo vykonat privilegovanou operaci. Navíc v mnoha částech systému není vhodně ošetřen výpis dat od uživatele, což umožňuje využít útoky typu Cross-site scripting.

4.6 Dokumentace

Dokumentace byla vytvořena ve formě materiálů, které se zabývají průběhem testování, tvorbou testů a testovacím prostředím. Všechny zde zmíněné materiály jsou umístěny ve složce s dokumentací na přiloženém CD.

Závěr

Cílem této práce bylo navrhnout a vytvořit prostředí vhodné pro testování již existujícího webového portálu vytvářeného pro podprou předmětu Databázové systémy. Práce se zabývala jednak stanovením vhodných postupů pro testování webových aplikací a jejich ověřením na vývojové verzi webového portálu, ale i automatizací celého procesu testovaní. Tyto cíle byly splněny.

Pro webový portál jsem navrhl vhodné testy, jejichž cílem bylo ověřit funkčnost portálu. Volil jsem především takové testy, které otestují portál jak z úrovně zdrojového kódu, tak i z pohledu běžného uživatele. Poté jsem portál podrobil navrženým testům a výsledky jsem popsal v kapitole 4.5.

V rámci práce jsem vytvořil také testovací prostředí, které jsem následně využil při realizaci průběžného testování. Mimoto jsem vytvořil také aplikaci, která umožňuje spustit automatické nasazení nové verze webového portálu do produkčního prostředí.

Nakonec jsem vytvořil podpůrné materiály, které nejenže popisují průběh testování a vytvořené testovací prostředí, ale obsahují také rady pro tvrorbu dalších testů.

Věřím, že tato práce poskytne vývojářům dobrý podklad pro budoucí testování celého systému.

Literatura

[1] Glazar, F.: *Systém pro podporu BI-DBS - semestrální práce*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

[2] Pejša, P.: *Systém pro podporu testování v BI-DBS*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

[3] Page, A.; Johnston, K.; Rollison, B.: *Jak testuje software Microsoft*. Brno: Computer Press, vyd. 1. vydání, 2009, ISBN 9788025128695, str. 209.

[4] Kota, K.: Testing Your Web Apps [online]. 2005, [vid. 28. 4. 2017]. Dostupné z: `https://www.adminitrack.com/articles/testing_web_apps.pdf`

[5] Bureš, M.; Renda, M.; Doležel, M.; aj.: *Efektivní testování softwaru*. Praha: Grada, první vydání, 2016, ISBN 978-80-247-5594-6, s. 39–56, 97, 99–103.

[6] Patton, R.: *Testování softwaru*. Praha: Computer Press, první vydání, 2002, ISBN 80-722-6636-5, s. 167–181, 186.

[7] Hlava, T.: Testy splněním a selháním [online]. 2011, [vid. 28. 4. 2017]. Dostupné z: `http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/testy-splnenim-a-selhanim`

[8] Hlava, T.: Fáze a úrovně provádění testů [online]. 2011, [vid. 4. 5. 2017]. Dostupné z: `http://testovanisoftwaru.cz/tag/systemove-testovani/#system`

[9] Nielsen, J.: Usability 101: Introduction to Usability [online]. 2012, [vid. 4. 5. 2017]. Dostupné z: `https://www.nngroup.com/articles/usability-101-introduction-to-usability/`

[10] Martin, R. C.: *Čistý kód*. Brno: Computer Press, vyd. 1. vydání, 2009, ISBN 9788025122853, str. 149.

[11] Ing. Ivan Halaška: Semestrální práce [online]. 2017, [vid. 4. 5. 2017]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-DBS/project/start

[12] Ing. Michal Valenta Ph.D.: Anotace [online]. 2017, [vid. 4. 5. 2017]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-DBS/annotation/start

[13] Jakub Jirůtka: OAuth 2.0 [online]. 2017, [vid. 8. 5. 2017]. Dostupné z: https://rozvoj.fit.cvut.cz/Main/oauth2

[14] Main - KOSapi - Projekt KOSapi [online]. 2017, [vid. 8. 5. 2017]. Dostupné z: https://kosapi.fit.cvut.cz/projects/kosapi/wiki

[15] Kubiš, M.: *Překladač z relační algebry do SQL*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

[16] Nette Tester – pohodové testování [online]. 2017, [vid. 8. 5. 2017]. Dostupné z: https://tester.nette.org/cs/

[17] Selenium IDE [online]. 2017, [vid. 8. 5. 2017]. Dostupné z: http://www.seleniumhq.org/projects/ide/

[18] Vagrant by HashiCorp [online]. 2017, [vid. 9. 5. 2017]. Dostupné z: https://www.vagrantup.com

[19] Docker - Build, Ship, and Run Any App, Anywhere [online]. 2017, [vid. 9. 5. 2017]. Dostupné z: https://www.docker.com

[20] Codeception [online]. 2017, [vid. 8. 5. 2017]. Dostupné z: https://codeception.com

[21] GitLab Continuous Integration & Deployment Pipelines | GitLab [online]. 2017, [vid. 9. 5. 2017]. Dostupné z: https://about.gitlab.com/features/gitlab-ci-cd/

[22] nette/code-checker: A simple tool to check source code against a set of Nette coding standards. [online]. 2017, [vid. 12. 5. 2017]. Dostupné z: https://github.com/nette/code-checker

[23] PSR-2: Coding Style Guide - PHP-FIG [online]. 2017, [vid. 12. 5. 2017]. Dostupné z: http://www.php-fig.org/psr/psr-2/

[24] slevomat/coding-standard: Slevomat Coding Standard for PHP_CodeSniffer extends Consistence Coding Standard by providing sniffs with additional checks. [online]. 2017, [vid. 12. 5. 2017]. Dostupné z: https://github.com/slevomat/coding-standard

[25] NEON sandbox [online]. 2017, [vid. 12. 5. 2017]. Dostupné z: https://ne-on.org

[26] Latte | Nette framework [online]. 2017, [vid. 12. 5. 2017]. Dostupné z: https://latte.nette.org/cs/

[27] fzaninotto/Faker: Faker is a PHP library that generates fake data for you [online]. 2017, [vid. 12. 5. 2017]. Dostupné z: https://github.com/fzaninotto/Faker

[28] Nextras components [online]. 2017, [vid. 12. 5. 2017]. Dostupné z: https://nextras.org/migrations/docs/3.0/

[29] Db - Codeception [online]. 2017, [vid. 12. 5. 2017]. Dostupné z: http://codeception.com/docs/modules/Db

[30] JakubOnderka/PHP-Parallel-Lint: This tool check syntax of PHP files faster than serial check with fancier output. [online]. 2017, [vid. 12. 5. 2017]. Dostupné z: https://github.com/JakubOnderka/PHP-Parallel-Lint

[31] squizlabs/PHP_CodeSniffer: PHP_CodeSniffer tokenizes PHP, JavaScript and CSS files and detects violations of a defined set of coding standards. [online]. 2017, [vid. 12. 5. 2017]. Dostupné z: https://github.com/squizlabs/PHP_CodeSniffer/

[32] phpstan/phpstan: PHP Static Analysis Tool - discover bugs in your code without running it! [online]. 2017, [vid. 12. 5. 2017]. Dostupné z: https://github.com/phpstan/phpstan

Příloha A

Seznam použitých zkratek

API Application Programming Interface

BI-DBS Databázové systémy

BI-SP1 Softwarový týmový projekt 1

BI-SP2 Softwarový týmový projekt 2

CSS Cascading Style Sheets

ČVUT České vysoké učení technické v Praze

HTML HyperText Markup Language

IDE Integrated Development Environment

MVC Model View Controller

NEON Nette Object Notation

PHP PHP: Hypertext Preprocessor

POST druh dotazovací metody Hypertext Transfer protokolu

PSR PHP Standards Recommendation

REST Representational State Transfer

SQL Structured Query Language

SRS Software Requirements Specification

URL Uniform Resource Locator

Příloha B

Kodovací standard

B.1 Hlavní pravidla

Tento code standard vychází z doporučení PSR-2[3] s několika rozdíly:

- Délka řádku není omezena.
- Současné používaní klíčových slov *else if* a *elseif* je povoleno.
- Klíčová slova *abstract* a *final* nemusí být před viditelností (*public*, *private* a *protected*) metody.

B.2 Další pravidla

Zatímco pravidla zmíněná v kapitole B.1 se zabývají formátováním zdrojového kódu, v této kapitole jsou uvedena další pravidla, která se zaměřují na „kvalitu" zdrojového kódu.

B.2.1 Pole

- Definice pole musí používat zkrácenou syntaxi *[]* místo *array()*.

B.2.2 Podmínky

- Podmínky nesmí mít prázdné tělo.
- Vyhodnocovací část podmínky nesmí být konstantní.

```
if (true) { # Chyba - konstantni podminka
    // ...
}
```

[3]Dostupné na http://www.php-fig.org/psr/psr-2

- Přiřazení v podmínce není povoleno.
- Yoda výrazy nejsou povoleny.

```
if (false == $smth) { # Chyba - Yoda vyraz
    // ...
}
```

- Používejte striktní porovnání === / !== místo == / !=.

B.2.3 Cykly

- Křížení parametrů v cyklech není povoleno.

```
for ($i = 0; $i < 1; $i++) {
    # Chyba - cyklus inkrementuje promennou $i
    for ($x = 0; $x < 1; $i++) {
        count([1,2]);
    }
}
```

- Cyklus nesmí v podmínce volat funkci.

```
# Chyba - podminka obsahuje funkci count()
for ($i = 0; $i < count([1,2]); $i++) {
    // ...
}
```

B.2.4 Soubory

- Soubor musí mít shodný název s názvem třídy uvnitř.
- Jmenný prostor musí být shodný s cestou k souboru. (PSR-4[4])

B.2.5 Třídy

- Třídy označené klíčovým slovem *final* nesmí obsahovat „finální“ metody.
- Třída nesmí obsahovat (override) metody, které pouze volají svého předka.
- Třída nesmí obsahovat *deprecated* funkce.
- Třída nesmí obsahovat funkce *sizeof* a *delete*.
- Třída nesmí obsahovat zbytečné *private* elementy.
- Konstanty musí mít uvedenou viditelnost (*public*, *private* a *protected*).

[4] Dostupné na `http://www.php-fig.org/psr/psr-4`

B.2.6 Jmenný prostor

- *Use* výrazy musí být seřazeny dle abecedy.
- Třída nesmí obsahovat nepoužívané *use* výrazy.
- *Use* výraz nesmí začínat na \.
- Třída nesmí obsahovat *use* výrazy ze stejného jmenného prostoru.

B.2.7 Type hinty

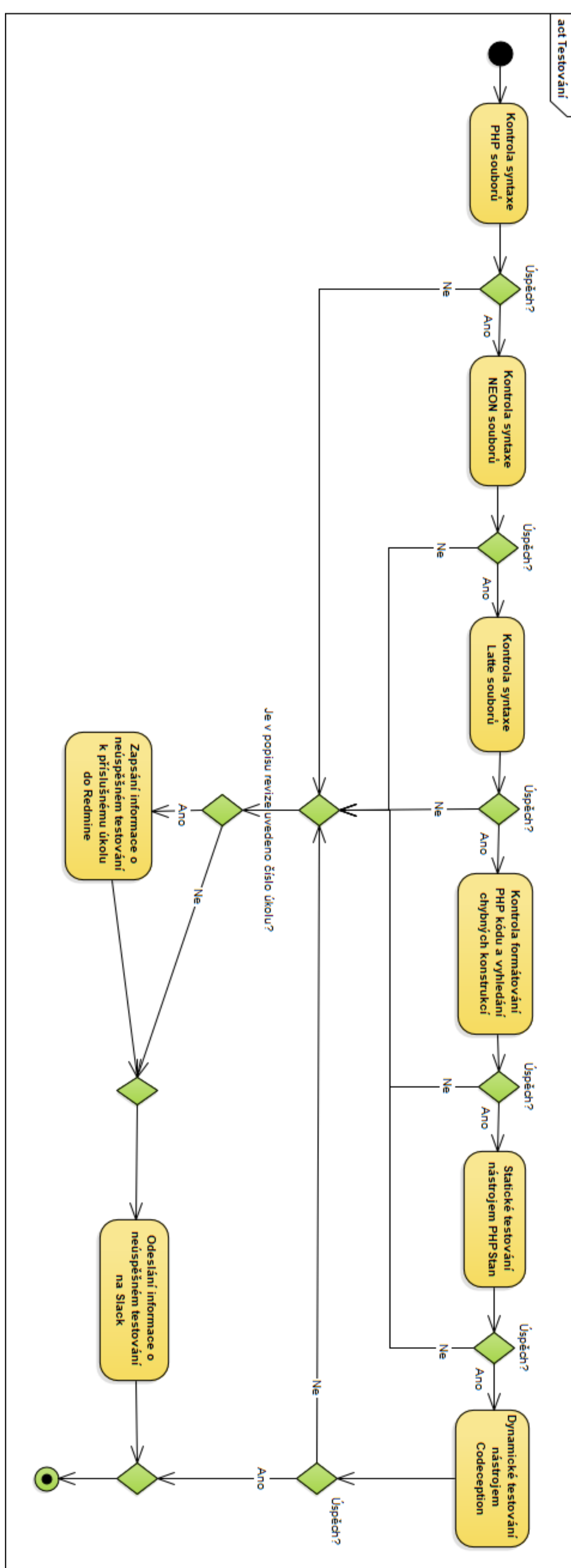
- U všech funkcí pište type hinty.
- Nullable type hint musí být psán dohromady *?string*.
- Proměnná, která má type hint a která má výchozí hodnotu *null*, musí mít nullable type hint.
- Type hinty musí být zapsané v „krátkém“ tvaru, například *integer* musí být zapsán jako *int*.

B.2.8 Ostatní

- Výraz *try-catch* nesmí obsahovat nedosažitelné *catch* bloky.
- K odchycení obecné výjimky použijte *\Throwable* místo *\Exception*.
- Používání superglobálních proměnných (*$_GET*, *$_POST*, ...) není povoleno.

Příloha C

Obrázky



Obrázek C.1: Proces průběžného testování

Příloha D

Obsah přiloženého CD

```
readme.txt ........................ stručný popis obsahu CD
docs .............................. adresář s dokumentací
  coverage .... protokol o pokrytí zdrojového kódu automatickými testy
  tests ........................... dokumentace testů
src
  project ......................... zdrojové kódy webového portálu a testů
  nette-module .................... zdrojové kódy modulu Nette
  environment zdrojové kódy skriptů pro sestavení virtuálních prostředí
  deploy-server ..... zdrojové kódy aplikace pro automatické nasazení
  thesis .......................... zdrojová forma práce ve formátu LaTeX
text .............................. text práce
  BP_Kovář_Pavel_2017.pdf ......... text práce ve formátu PDF
images ............................ složka s UML diagramy
```