UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2018 Martin Volenec

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Testovatelná webová aplikace typu e-shop

Martin Volenec

Bakalářská práce

2018

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin Volenec**
Osobní číslo:	**I14200**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Testovatelná webová aplikace typu e-shop**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem bakalářské práce je návrh a implementace webové aplikace (typu e-shop) s využitím jednotkových testů. Aplikace bude vybudována na platformě ASP.NET s využitím ORM frameworku Entity.
V teoretické části práce budou popsány zvolené technologie pro vytvoření webové aplikace. Dále bude proveden teoretický úvod do jednotkových testů a problematiky jejich použití při realizaci webové aplikace s využitím zvoleného architektonického modelu (např. MVC, MVP, MVVM, ...).
V praktické části bude navržena a implementována webová aplikace. Pro aplikaci bude rovněž vytvořena sada jednotkových testů, volitelně dále integračních testů, které otestují funkčnost modelu a řídící části webové aplikace.

Rozsah grafických prací:

Rozsah pracovní zprávy: **40 stran**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

FREEMAN, Adam. Pro ASP.NET MVC 5. 5th ed. New York, N.Y.: Apress, 2013. ISBN 9781430265306.
NAYLOR, Lee. ASP.NET MVC with Entity Framework and CSS. United Kingdom: Apress, 2016. ISBN 9781484221372.
MCWHERTER, Jeff a Ben HALL. Testing ASP.NET Web Applications. Indianapolis, Indiana: John Wiley, 2011. ISBN 9781118081228.
MVC Recommended Resources. Microsoft Docs [online]. 2014 [cit. 2017-10-19]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/recommended-resources-for-mvc

Vedoucí bakalářské práce: **Ing. Roman Diviš**
Katedra softwarových technologií

Datum zadání bakalářské práce: **31. října 2017**

Termín odevzdání bakalářské práce: **12. května 2018**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

Ing. Lukáš Čegan, Ph.D.
pověřený vedením katedry

V Pardubicích dne 20. března 2018

Prohlášení autora

Prohlašuji, že jsem tuto práci vykonal samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména ze skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 07. 05. 2018

Martin Volenec

PODĚKOVÁNÍ

Zde bych chtěl poděkovat panu Ing. Romanu Divišovi za strávený čas a cenné rady při zpracování bakalářské práce. Hlavně za rychlé odezvy při dotazech a revizích. Dále bych chtěl poděkovat celé své rodině a přítelkyni za poskytnutou oporu během celého studia.

ANOTACE

Práce se zaměřuje na realizaci testovatelné aplikace typu e-shop s využitím architektonického modelu MVC. Pro vývoj byly využity technologie ASP.NET a Entity framework. Dále bylo naimplementováno 40 jednotkových testů a byla vysvětlena problematika izolace jednotkových testů.

KLÍČOVÁ SLOVA

E-shop, webová aplikace, Entity framework, C# ASP .NET, MVC, jednotkové testy, Moq

TITLE

Testable web-based e-shop application

ANNOTATION

The work focuses on the realization of a tested e-shop application using the architectural model of MVC. For development, ASP.NET and Entity Framework were used. In addition, 40 unit tests were performed and the issue of isolation of the unit tests was explained.

KEYWORDS

E-shop, web application, Entity framework, C# ASP .NET, MVC, unit tests, Moq

OBSAH

ÚVOD 13

1 Popis použitých technologií 14

1.1 C# 14

1.2 ASP.NET 14

1.3 Entity Framework 14

1.4 MVC 15

1.5 MS Test 16

1.6 Moq 16

2 Testování aplikace 17

2.1 Jednotkové testy 17

2.2 Integrační testy 21

2.3 Systémové testy 21

2.4 Akceptační testy 21

3 Problém izolace jednotkových testů 22

3.1 Fake 22

3.2 Stub 23

3.3 Spy 24

3.4 Mock 24

3.5 Moq 25

4 Implementace aplikace 27

4.1 Problém návrhu aplikace 28

4.2 Řešení pomoci Dependency injection 30

4.3 Testování datové vrstvy 31

4.4 Testování business vrstvy 33

5 Závěr 36

Použitá literatura 37

Seznam Příloh 39

SEZNAM ILUSTRACÍ A TABULEK

Obrázek 1 - Architektura MVC. Zdroj: [16].....15
Obrázek 2 - Postup spuštění testu pomocí atributu [TestMethod].....20
Obrázek 3 - Spuštění všech testů 20
Obrázek 4 - Výsledky testů.....20
Obrázek 5 - Diagram objektu typu Fake. Zdroj: [17].....23
Obrázek 6 - Diagram objektu typu Stub. Zdroj: [18] 24
Obrázek 7 - Diagram objektu typu Mock. Zdroj: [19] 25

SEZNAM ZDROJOVÝCH KÓDŮ

Zdroj. kód 1 - Jednotkový test s použitím techniky AAA 18
Zdroj. kód 2 - Základní jednotkový test v MS Tests 18
Zdroj. kód 3 - Jednotkový test MS Tests s použitím atributu [TestInitialize] 19
Zdroj. kód 4 - Jednotkový test s využitím atributu [ExpectedException] 20
Zdroj. kód 5 - Rozhraní ISendable 25
Zdroj. kód 6 - Vytvoření Moqu, který vrací true 26
Zdroj. kód 7 - Vytvoření Moqu, který vrací false 26
Zdroj. kód 8 - Využití metody Verify 26
Zdroj. kód 9 - Použití bezparametrického konstruktoru 28
Zdroj. kód 10 - Settery pro Moq objekty 29
Zdroj. kód 11 - Úspěšné nastavení instance Moq objektu 29
Zdroj. kód 12 - Rozhraní ICompanyRepository 30
Zdroj. kód 13 - Manažer využívající Dependency Injection 30
Zdroj. kód 14 - Předání instance Moqu pomocí Dependency Injection 30
Zdroj. kód 15 - Test metody FindByUrl 31
Zdroj. kód 16 - Test metody FindByCategoryId 31
Zdroj. kód 17 - Test metody FindProductById 32
Zdroj. kód 18 - Práce s cookies a session před úpravou 33
Zdroj. kód 19 - Obalová třída pro práci s cookies a session 34
Zdroj. kód 20 - Metoda CreateOrder s využitím BusinessHelperu 35
Zdroj. kód 21 - Získání objednávky ze session při testování 35

SEZNAM ZKRATEK A ZNAČEK

AAA	Arrange, Act, Assert
ASP	Active Server Pages
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
ISS	Internet Information Services
JSON	Java Script Object Notation
JSP	Java Server Pages
MVC	Model-view-controller
OOP	Objektově orientované programování
ORM	Objektově relační mapování
PHP	Hypertext Preprocessor
SQL	Structured Query Language
URL	Uniform Resource Locator
XML	Extensible Markup Language

ÚVOD

V dnešní době, kdy klasické nakupování a kamenné obchody pomalu zanikají, se obchod přesunul na internet a aplikace typu e-shop nabývají na své důležitosti a oblíbenosti. Díky e-shopům můžeme nakupovat z pohodlí domova a díky čím dál lepších služeb dopravních společností obdržíme zboží druhý den. V některých případech již v den objednání.

Pro běžného zákazníka je e-shop velmi jednoduchá služba. Vybere si zboží, zaplatí a poté čeká na doručení zboží. Nicméně e-shop není triviální aplikací z ohledu na rozsah, databázi, potřebné funkcionality a na právní hledisko. Tato bakalářská práce se zabývá návrhem a tvorbou testovatelné webové aplikace typu e-shop. V rámci této práce je vypracovaná analýza a návrh webové aplikace typu e-shop. Základním nefunkčním požadavkem na tuto aplikaci je doplnit její zdrojové kódy o sadu testů pro ověření správné funkčnosti.

Testovat by se měla každá aplikace. Testy umožňují ověřit správnou funkčnost aplikace za různých podmínek. U tohoto typu aplikace lze očekávat dynamický vývoj i po prvním nasazení aplikace, e-shop je nutné udržovat v souladu s legislativou a také doplňovat nové funkce s ohledem na zákazníka. A i přes to, že nová funkcionalita je při vývoji otestována, může se stát, že na produkční větvi projektu to fungovat nebude. Poté e-shop nefunguje a majitel přichází o peníze a zákazníky. To je situace, která je nežádoucí. Díky testům je možné pokusit se minimalizovat výskyt problémů, testování rovněž umožňuje včas detekovat chyby v průběhu vývoje aplikace a předchází zpoždění a podražení vývoje aplikace. Mezi základní kategorie softwarových testů lze zařadit testy jednotkové, integrační, nebo akceptační. Tato práce se zaměřuje zejména na uplatnění jednotkových testů (tzv. unit tests).

1 POPIS POUŽITÝCH TECHNOLOGIÍ

Účelem této kapitoly je popis technologií použitých při vývoji této práce.

1.1 C#

C#, který se anglicky vyslovuje jako C Sharp, je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou *Microsoft*. Byl vyvinut zároveň s platformou *.NET Framework*, který má své kořeny taktéž v roce 2000. *C#* je inspirován jazyky *C++* a *Java*.

C# lze využít k tvorbě databázových programů, webových aplikací a stránek, webových služeb, formulářových aplikací ve *Windows*, softwaru pro mobilní zařízení atd.

1.2 ASP.NET

ASP.NET je webový framework, který umožňuje tvorbu webových aplikací v jazyce *C#*. Díky tomu, že *ASP.NET* je součástí *.NET frameworku*, najdeme zde již mnoho řešení základních problémů, které ve webových technologií nastávají [9] [1]. Tato technologie je nástupcem *ASP (Active Server Pages)*. Její přímá konkurence je *JSP (Java Server Pages)*.

ASP.NET je založen na architektuře klient-server. Aplikace typu *ASP.NET* je tedy program, který běží na straně serveru [8]. Na základě požadavků od klienta vygeneruje webovou stránku a pošle ji klientovi. Ten vidí už jenom výsledné *HTML*.

ASP.NET běží na serveru *IIS (Internet Information Services)*. Ale existují moduly pro webové servery *Apache* a *Nginx*.

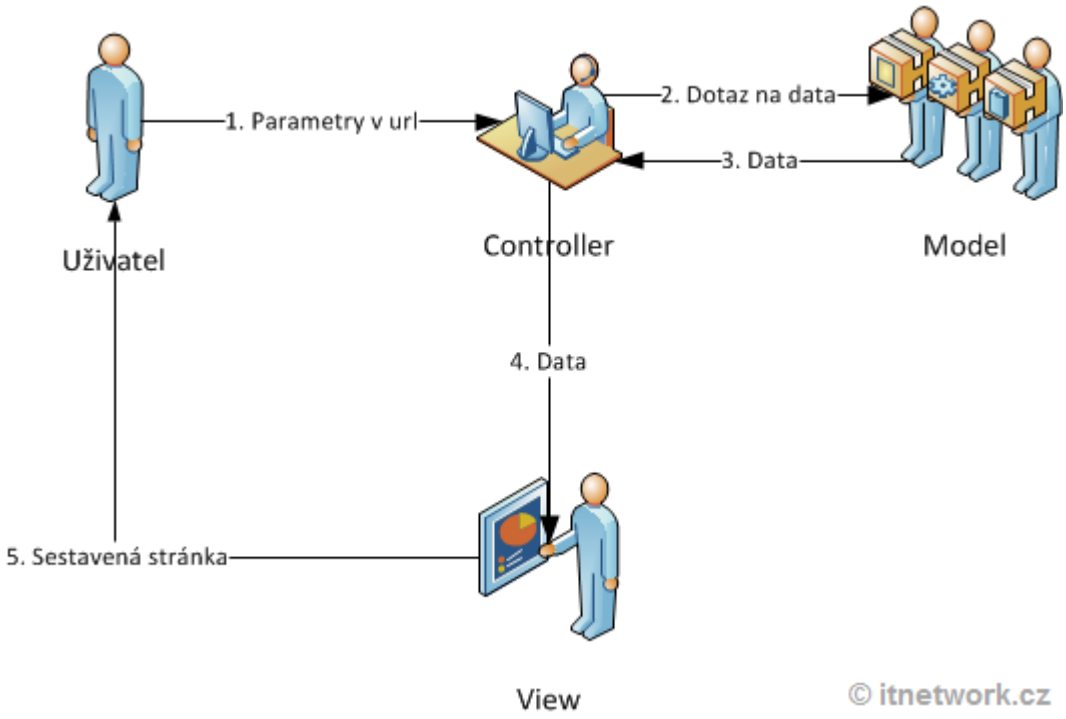
1.3 Entity Framework

Entity Framework představuje *ORM* framework. Realizuje objektově-relační mapování mezi databází a objekty v jazyce *C#* [2]. Tudíž v kódu se pracuje jen s objekty a framework sám na pozadí generuje a provádí *SQL* dotazy. S jazykem *SQL* není nutné pracovat a aplikace je 100 % objektová. *Entity Framework* byl původně součástí *.NET Frameworku*, ale od verze 6 je veden jako samostatný projekt.

Entity Framework nabízí dva způsoby, jak přistoupit k tvorbě projektu [6]. První se jmenuje Database First. Nejdříve je třeba založit databázi a poté nám *Entity Framework* vygeneruje třídy a kontext. Druhý způsob se nazývá Code First. Pomocí jazyka *C#* se nejdříve vytvoří třídy a *Entity Framework* nám podle nich automaticky vygeneruje databázi a potřebný kontext. Tento způsob je mnohem jednodušší a oblíbenější [2].

1.4 MVC

Model-View-Controller je softwarová architektura používaná pro návrh komplexních aplikací. Tato architektura rozděluje řídící logiku, uživatelské rozhraní a datový model aplikace do tří nezávislých komponent [7]. Díky tomu modifikace některé komponenty má jen minimální vliv na ostatní komponenty a naše aplikace se stává mnohem přehlednější. Mnoho dnešních populárních frameworků, jako například *Zend*, *Symphony*, *Ruby On Rails* nebo právě *ASP.NET*, na této architektuře stavějí.



Obrázek 1 - Architektura MVC. Zdroj: [16]

Základní komponenty tvoří Model, View a Controller.

Model je komponenta, která obsahuje logiku a vše, co do ní spadá. Lze si pod tím představit výpočty, databázové dotazy, validace dat a podobně. Důležité je, že model neví o výstupu, odkud mu data v parametrech přišla a co se s nimi bude dít dále. Jeho funkce spočívá pouze v tom přijmout parametry a vydat data.

View má jen jednu úlohu. Musí se postarat o to, aby zobrazil výstup uživateli. Výstupem může být cokoliv. Například *HTML*, *XML* nebo *JSON*.

Controller je prvek, který funguje jako jakýsi prostředník. Komunikuje s ním uživatel, Model a View [7]. Controller všechny tyto komponenty propojuje a předává mezi nimi požadované informace.

Životní cyklus této architektury funguje následujícím způsobem (viz obrázek 1). Na začátku je uživatel, který přistoupí na webovou stránku a vytvoří tak *HTTP* požadavek s parametry v *URL*. Jako první zachytí požadavek Router. Úloha routeru je správně přeložit *URL* a zavolat příslušný Controller. Pokud parametry projdou validací, tak Controller vystaví dotaz na data Modelu. Poté, co Model vrátí data, tak je Controller předá View. A View vrátí již sestavenou stránku, kterou vidí uživatel.

1.5 MS Test

MS Test je framework, který je součástí *IDE Visual Studio*. Byl vyvinut společností *Microsoft*. Slouží ke psaní jednotkových testů.

1.6 Moq

Moq objekty umožňují napodobit chování tříd a rozhraní a nechat kód v testu s nimi pracovat, jako by byly skutečné. Objekt typu Moq lze kompletně nastavit. A to včetně zadávání parametrů a návratových hodnot. Lze taktéž ověřit, zda jsou v testovaném kódu volány metody, které mají být volány. Tím je zajištěno, že tok programu je podle očekávání.

2 TESTOVÁNÍ APLIKACE

Úkolem testování aplikací je zajistit kvalitu výsledného produktu a jeho shodu se zadáním. Velmi často se testování omezuje a zanedbává, což může zdánlivě přinést úsporu prostředků, ale dlouhodobě může mít velmi negativní účinky na vztah se zákazníkem a v konečném důsledku i na celkové náklady [15]. Díky pokrokům v hardwarových a softwarových technologiích se dá efektivita testování výrazně zvýšit.

2.1 Jednotkové testy

Jednotkové testy slouží k otestování jednotlivých, dále již nedělitelných, jednotek zdrojového kódu. Jednotkou se v *OOP* běžně rozumí metoda. Testují třídy, přesněji jejich metody. Předávají jim různé vstupy a zkouší, zda jsou jejich výstupy korektní. Díky tomu máme ověření, že kód funguje tak, jak očekáváme. Testy jsou vykonávány sériově. Test je ve stavu passed/fail. Pokud selže jediný test z celé sady testů, tak i celá sada testů je ve stavu fail.

Základními požadavky na jednotkový test jsou:

- Krátký – jednotkový test může být typicky realizován na 3 řádcích kódu.
- Rychlý – provedení jednotkového testu by mělo být kratší než jedna sekunda.
- Deterministický – vždy ze stejných vstupních podmínek svým během vytvoří stejné výsledky. Je tedy předvídatelný.
- Neobsahuje logiku (cykly, podmínky).
- Nezávislý na prostředí a okolních komponentách.

Díky jednotkovým testům lze bezpečně provádět opravy a refactoring zdrojového kódu. Pokud je kód pokryt jednotkovými testy, dojde ke zjištění, zda došlo k poruše jiné části kódu [12]. Následně na základě těchto testů lze stanovit tzv. *Code coverage*. Jedná se o měření, kolik řádků kódu se provádí během vykonávaní automatizovaných testů a tím sděluje množství pokrytí kódu testy.

Při vytváření jednotkových testů by se mělo pracovat s technikou *AAA (arrange, act, assert)*. Jedná se o velmi důležitou a základní myšlenku jednotkového testování. Jednotkový test se skládá ze tří kroků.

- **Arrange** – Jedná se o první krok testování jednotky. Provádí se zde potřebné nastavení testu. Slouží například k inicializaci testovaného objektu.
- **Act** – Ve druhém kroku je proveden test. Volání cílené funkce pomocí objektu, který byl vytvořen v předchozím kroku.

- **Assert** – Poslední krok testu jednotky slouží ke kontrole a ověření vráceného výsledku s očekávanými výsledky.

```
[TestMethod]
public void TestMethod()
{
       //Arrange
       TestClass objTest = new TestClass();

       //Act
       var result = objTest.testFunction();

       //Assert
       Assert.AreEqual(true, result);
}
```

Zdroj. kód 1 - Jednotkový test s použitím techniky AAA

Důležitou částí jednotkového testu je použití třídy Assert. V této třídě se nacházejí metody, které ověří výstup testované funkce a rozhodnou, zda byl jednotkový test úspěšný.

Skoro každý programovací jazyk má vlastní framework pro jednotkové testy. Jazyk *Java* pracuje s *JUnit*, *PHP* má *PHPUnit*. Jednotkové testy v jazyce *C#* lze napsat pomocí více frameworků. Na výběr jsou tyto možnosti:

- *MS Tests*,
- *NUnit*,
- *XUnit*.

Všechny tyto frameworky mají podobný cíl. Psát jednotkové testy rychle a jednoduše. Ale je mezi nimi pár klíčových rozdílů. Pro tuto práci je použit framework *MS Test*.

```
[TestClass]
public class CalculatorTests
{
    [TestMethod]
    public void CalculatorSumTest()
    {
        Calculator calculator = new Calculator();

        int result = calculator.sum(1, 1);

        Assert.AreEqual(2, result);
    }
}
```

Zdroj. kód 2 - Základní jednotkový test v MS Tests

Zdrojový kód č. 2 zobrazuje jednoduché vytvoření jednotkového testu v *MS Tests*. Pro základní a pokročilou práci s tímto frameworkem stačí pět atributů a třída Assert:

- [TestClass]
- [TestInitialize]
- [TestMethod]
- [TestCleanup]
- [ExpectedException]

Atribut [TestClass] slouží k tomu, aby *MS Test* pracovalo s touto třídou jako s třídou testovací. A [TestMethod] slouží k označení každé metody, ze které má být vytvořen jednotkový test.

Další atributy umožňují zefektivnit proces vytváření jednotkových testů, které mají častokrát podobný základ a před jejich vyvoláním je nutné připravit testovaný či testující objekt [13]. Metoda označena atributem [TestInitialize] se vykoná před každým spuštěním jednotkového testu. Tato metoda může sloužit například k vytvoření testovacích objektů a dat. Obdobou je metoda označená atributem [TestCleanup]. Taková metoda je automaticky volána po každém dokončení jednotkového testu. Například pro dealokaci vytvořených objektů.

```
[TestClass]
public class CalculatorTests
{
    private Calculator calculator;

    [TestInitialize]
    public void Initialize()
    {
        calculator = new Calculator();
    }

    [TestMethod]
    public void CalculatorSumTest()
    {
        int result = calculator.sum(1, 1);

        Assert.AreEqual(2, result);
    }
}
```

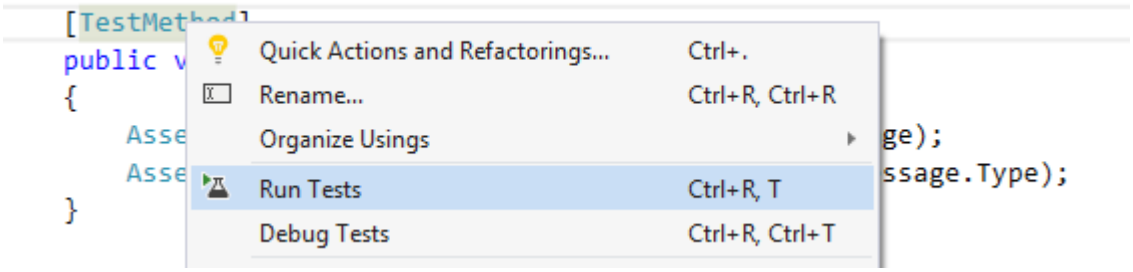
Zdroj. kód 3 - Jednotkový test MS Tests s použitím atributu [TestInitialize]

Pokud testovaná metoda musí vyvolat za určitých podmínek danou výjimku, požije se atribut [ExpectedException] spolu s [TestMethod]. Pokud bude výjimka vyvolána, test projde jako úspěšný.

```
[TestMethod]
[ExpectedException(typeof(IndexOutOfRangeException))]
public void TestArrayException()
{
    int[] numbers = new int[5];
    numbers[5] = 10;
}
```

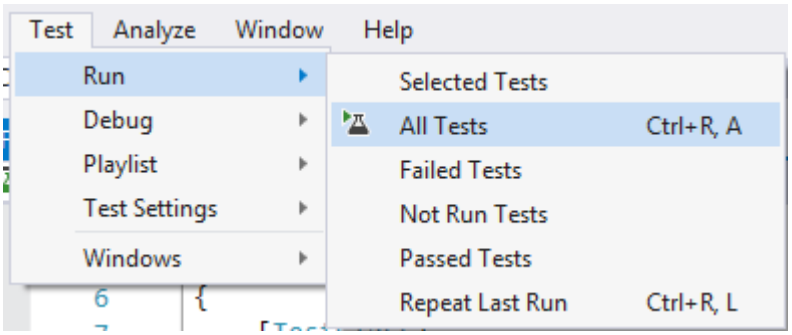
Zdroj. kód 4 - Jednotkový test s využitím atributu [ExpectedException]

Díky tomu, že framework *MS Test* je součástí *IDE Visual Studio*, tak spuštění testů je velmi jednoduché. Stačí pravým tlačítkem kliknout na klíčové slovo [TestMethod] a zvolit Run Tests.

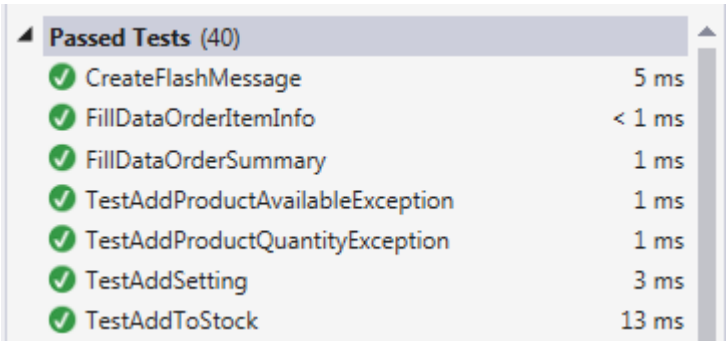


Obrázek 2 - Postup spuštění testu pomocí atributu [TestMethod]

Ke spuštění všech jednotkových testů najednou slouží hlavní horní menu s nabídkou Test. Poté nabídka Run a All Tests. Po provedení testů *Visual Studio* nabídne přehled, jak testy dopadly.



Obrázek 3 - Spuštění všech testů



Obrázek 4 - Výsledky testů

2.2 Integrační testy

Pro testování více komponent dohromady slouží integrační testy. Musí být ověřena bezchybná komunikace mezi jednotlivými komponentami uvnitř aplikace. Integraci lze však ověřovat nejen mezi komponentami, ale také mezi komponentou a operačním systémem, hardwarem či rozhraním různých systémů [12]. V této fázi se tak testuje integrace dosud jednotlivě ověřených částí. Postupně probíhá testování integrace mezi dvěma komponentami a pak se přidávají další. Integrační testy mohou být jak manuální, tak i automatizované.

2.3 Systémové testy

Po ověření správné integrace je možné pokračovat na další úroveň testů, tj. na 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 testovány [12]. Je to tedy poslední úroveň testů, které se provádějí před předáním produktu zákazníkovi pro akceptační testování. Většinou slouží jako výstupní kontrola aplikace. Bez této úrovně by celé testování aplikace nemělo význam. Bezporuchovost výsledné aplikace by byla významně ohrožena. Proto tuto úroveň testů můžeme považovat za stěžejní v celém postupu testování aplikace.

2.4 Akceptační testy

User acceptance tests (*UAT*) jsou akceptační testy na straně zákazníka. Pokud všechny předchozí etapy testů proběhly bez větších nedostatků, je možné předat aplikaci zákazníkovi. Je to v podstatě přejímací zkouška aplikace [12]. Zákazník si většinou se svým týmem testerů provede akceptační testy. Ty jsou často prováděny podle připravených scénářů, které společně připravil zákazník s dodavatelem aplikace. Testy probíhají na testovacím prostředí u zákazníka. Nalezené nesrovnalosti mezi aplikací a specifikací jsou ihned reportovány vývojovému týmu. Ty jsou v pak dohodnuté lhůtě odstraněny a nasazeny na prostředí u zákazníka.

Tato poslední část testování aplikace lze zautomatizovat. Slouží k tomu testovací framework *Selenium* [11]. To je nástroj pro automatické testování webových aplikací. Díky němu lze simulovat reálného uživatele a testuje tak z pohledu uživatele, jestli funguje celý web.

3 PROBLÉM IZOLACE JEDNOTKOVÝCH TESTŮ

Ačkoliv jsou jednotkové testy velmi efektivní, tak i přes to mají svá omezení. Pokud dojde k testování části kódu, kde se pracuje se závislostí na prostředí nebo databázi, tak vyřešit tuto situaci pomocí jednotkových testů je velmi složité [15]. Proto se při automatizovaném testování běžně používají objekty, které vypadají a chovají se jako jejich ekvivalenty na produkční větvi. Ale ve skutečnosti jsou zjednodušené. To umožňuje ověřit kód nezávisle na zbytku systému. Obecný termín používaný pro tyto objekty je *Test Doubles*.

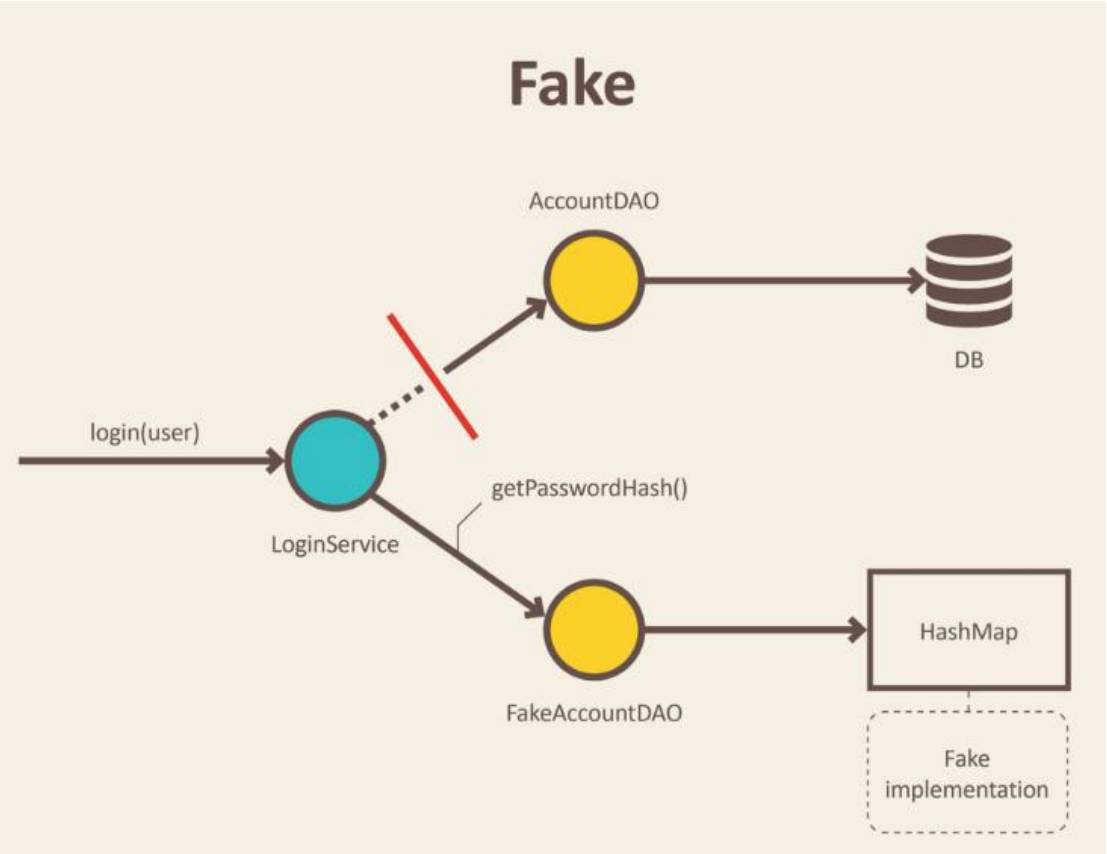
V této části bakalářské práce budou popsány tyto varianty implementace *Test Doubles*. Jsou jimi:

- *Fake*,
- *Stub*,
- *Spy*,
- *Mock*.

3.1 Fake

Fake jsou objekty, které mají fungující implementaci, ale ne stejnou jako na produkční větvi [14]. Obvykle obsahují zkrácenou a zjednodušenou verzi kódu.

Příkladem může být implementace repositáře v paměti počítače. Tedy třídy, která pracuje s databází. Tato falešná implementace nebude využívat databázi, ale bude používat jednoduchou kolekci pro ukládání dat. To nám umožní provést integrační test bez spuštění databáze a provádění časově náročných požadavků. Lze tedy rychle implementovat a provozovat systém v paměti počítače a odkládat rozhodnutí o návrhu databáze.

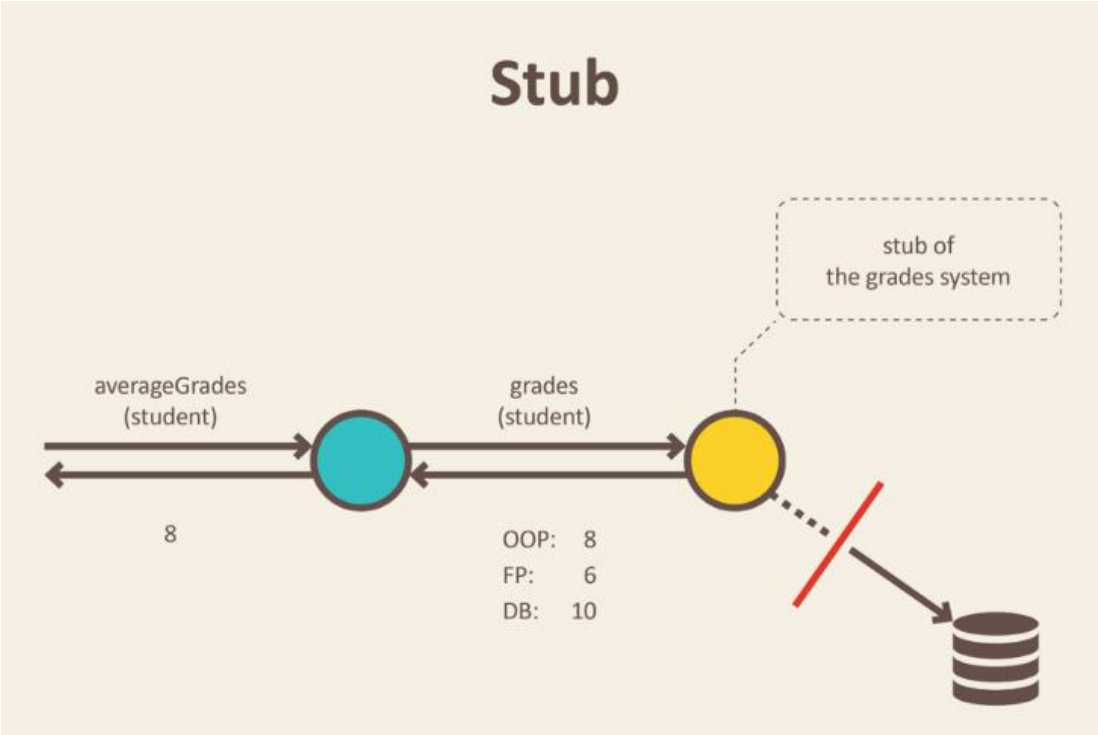


Obrázek 5 - Diagram objektu typu Fake. Zdroj: [17]

3.2 Stub

Stub je objekt, který obsahuje předdefinovaná data a používá jej během testů. Používá se, když nelze zahrnovat objekty, které by odpovídaly reálnými daty nebo by měly nežádoucí vedlejší účinky.

Příkladem může být objekt, který potřebuje získat data z databáze. Namísto skutečného objektu se vytvoří stub a definuje se, jaké údaje by měly být vráceny.



Obrázek 6 - Diagram objektu typu Stub. Zdroj: [18]

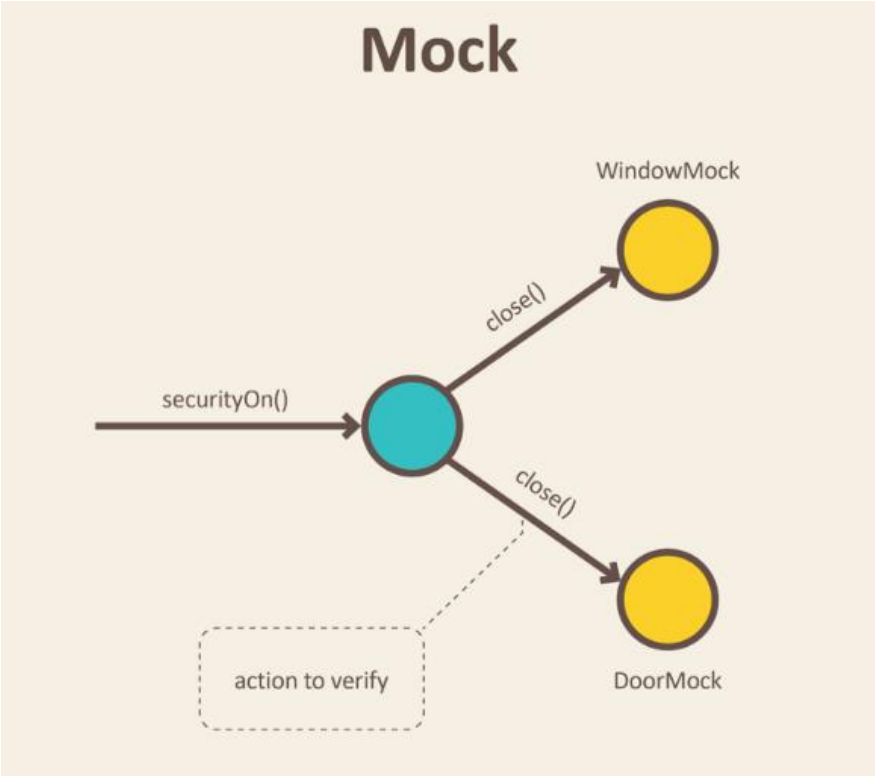
3.3 Spy

Spy je objekt, který se chová jako *Stub*, ale také zaznamenává každé jeho volání a může tedy ověřit, že určitá interakce proběhla dle plánu. Jejich nejčastější užití je, když testovaný subjekt spouští vedlejší efekt, který nemůže být potvrzen návratovou hodnotou.

3.4 Mock

Mock je objekt, který registruje volání, která obdrží. Při provádění testu provede ověření, zda na mock objektu byly provedeny všechny očekávané akce [14].

Mock lze využít v případě, když neexistuje jednoduchý způsob ověření, že zamýšlený kód byl proveden [14]. Když není k dispozici návratová hodnota nebo žádný způsob, jak zkontrolovat změnu stavu systému. Příkladem může být funkce, která volá službu na posílání e-mailů. Nelze posílat e-maily pokaždé, když je test spuštěn. Kromě toho není snadné ověřit, že byl poslán správný e-mail. Jediné, co lze udělat, je ověřit, zda byla funkce na odesílání e-mailů zavolána.



Obrázek 7 - Diagram objektu typu Mock. Zdroj: [19]

3.5 Moq

Mock objekty lze v *C#* realizovat například pomocí open-source .NET frameworku Moq (https://github.com/moq/moq4), který byl v této práci použit. Poskytuje jednoduchou a přímočarou práci s objekty typu *Mock*.

Zdrojový kód č. 5 představuje rozhraní, které implementuje třída, která se stará o odesílání e-mailů. Pokud se e-mail korektně odešle, metoda vrátí hodnotu true, jinak false. Následně by se testovala část kódu, kde se s tímto objektem pracuje. Pouze pomocí jednotkového testu není možno tuto funkcionalitu otestovat, neboť o odesílání e-mailu se stará server. Je potřeba tuto situaci zjednodušit pomocí objektu *Mock*.

```
public interface ISendable
{
    bool Send();
}
```

Zdroj. kód 5 - Rozhraní ISendable

```
[TestMethod]
public void TestOrderMailSent()
{
        // ...
        Mock<ISendable> mailerMock = new Mock<ISendable>();
        mailerMock.Setup(x => x.Send()).Returns(true);
}
```

Zdroj. kód 6 - Vytvoření Moqu, který vrací true

Zdrojový kód č. 6 zobrazuje vytvoření objekt typu *Mock*, který je generický. Lze použít rozhraní, který simulovaný objekt implementuje, nebo přímo konkrétní třídu. Při použití tohoto mock objektu v testování se nebude odesílat e-mail, ale mock vrátí hodnotu true. Tím bude nasimulováno odeslání e-mailu a účel mocku bude splněn.

K testování opačného chování, tedy neúspěšného odeslání e-mailu, by kód vypadal následovně:

```
[TestMethod]
public void TestOrderMailNotSent()
{
        // ...
        Mock<ISendable> mailerMock = new Mock<ISendable>();
        mailerMock.Setup(x => x.Send()).Returns(false);
}
```

Zdroj. kód 7 - Vytvoření Moqu, který vrací false

K ověření kolikrát byla metoda zavolána slouží metoda Verify. Jako první argument je metoda, na kterou se ptáme. Do druhého se vkládá hodnota, kolikrát měla být zavolána. K tomu slouží třída Times, která je součástí Moq frameworku.

```
[TestMethod]
public void TestOrderMailSent()
{
        // ...
        Mock<ISendable> mailerMock = new Mock<ISendable>();
        mailerMock.Setup(x => x.Send()).Returns(true);

        // ...
        mailerMock.Verify(x => x.Send(), Times.Once());
}
```

Zdroj. kód 8 - Využití metody Verify

4 IMPLEMENTACE APLIKACE

Pro implementaci testování slouží webová aplikace typu e-shop. E-shop slouží k nabídce a vyhledání zboží s možností objednávky, příjmu objednávek od zákazníka a zprostředkování. Dále nabízí možnost zboží vyhledat podle parametrů či podle shody v názvu. Všechny jednotlivé produkty mají detailní stránku s podrobným popisem, která je generována dynamicky pomocí dat z databáze. Uživatelům je umožněno procházet zboží, které je organizováno v kategoriích. Zboží si může objednat i neregistrovaný uživatel. Správce e-shopu po přihlášení má k dispozici základní menu pro základní správu e-shopu. Vytváření nových produktů včetně zařazení do kategorií a následně jejich správu. Pro front-end e-shopu byl použit framework Bootstrap 3. E-shop ukládá uživatele zboží jednak do databáze, ale taktéž využívá mechanismus, který uloží token do *cookies* prohlížeče a pokud uživatel mezitím *cookies* nesmaže, při návratu bude mít uživatel stejný košík jako při poslední návštěvě.

Aplikaci je rozdělena do tří vrstev.

- **Datová vrstva** – Obsahuje část, která se stará o ukládání a načítání dat. Ostatní části využívají její metody, aniž by se zajímaly o to, kde jsou data uložena. Kód pro práci s daty je implementován v této vrstvě v podobě repositářů. ER diagram databáze je k dispozici jako příloha A.
- **Business vrstva** – Obsahuje téměř všechnu logiku naší aplikace rozdělenou do tzv. manažerů. Dále se zde nacházejí pomocné třídy.
- **Aplikační vrstva** – Obsahuje controllery, pohledy a části, které patří k webové aplikaci. CSS styly, Javascript a obrázky.

4.1 Problém návrhu aplikace

Při testování aplikace vznikl problém, jak dostat Moq objekty do testovaných tříd.

```
public class OrderManager : IOrderManager
{

    private IOrderRepository orderRepository;
    private IProductRepository productRepository;
    private IProductOrderRepository productOrderRepository;
    private ICategoryRepository categoryRepository;
    private IAccountingSettingManager accountMannager;
    private IBusinessHelper businessHelper;

    public OrderManager()
    {
        this.orderRepository = new OrderRepository();
        this.productRepository = new ProductRepository();
        this.productOrderRepository = new ProductOrderRepository();
        this.categoryRepository = new CategoryRepository();
        this.accountMannager = new AccountingSettingManager();
        this.businessHelper = new BusinessHelper();
    }

}
```

Zdroj. kód 9 - Použití bezparametrického konstruktoru

Manager standardně využívá bezparametrický konstruktor, který automaticky vytváří jednotlivé objekty tříd, na kterých je závislý. Není tedy možné tyto objekty nahradit. Řešením je přidání setterů pro všechny repositáře a manažery. Avšak toto řešení není plně korektní, neboť porušuje jedno z pravidel testování. Lze změnit kód jednotkového testu, ale aplikační kód se nesmí změnit pro test jednotky. A tato situace nastala.

```
public class OrderManager : IOrderManager
{

    private IOrderRepository orderRepository;
    private IProductRepository productRepository;
    private IProductOrderRepository productOrderRepository;
    private ICategoryRepository categoryRepository;
    private IAccountingSettingManager accountMannager;
    private IBusinessHelper businessHelper;

    public OrderManager()
    {
        this.orderRepository = new OrderRepository();
        this.productRepository = new ProductRepository();
        this.productOrderRepository = new ProductOrderRepository();
        this.categoryRepository = new CategoryRepository();
        this.accountMannager = new AccountingSettingManager();
        this.businessHelper = new BusinessHelper();
    }

    public void SetOrderRepository(IOrderRepository repository)
    {
        this.orderRepository = repository;
    }

    public void SetProductRepository(IProductRepository repository)
    {
        this.productRepository = repository;
    }

     // ...

}
```

Zdroj. kód 10 - Settery pro Moq objekty

Nyní lze jednoduše v jednotkovém testu předat instanci Moq objektu.

```
[TestMethod]
public void TestCreateOrder()
{
    Mock<IOrderRepository> orderRepositoryMock = new Mock<IOrderRepository>();
    Mock<IBusinessHelper> businessHelperMock = new Mock<IBusinessHelper>();

    var orderManager = new OrderManager();
    orderManager.SetOrderRepository(orderRepositoryMock.Object);
    orderManager.SetBusinessHelper(businessHelperMock.Object);

    Assert.IsNotNull(orderManager.CreateOrder());
}
```

Zdroj. kód 11 - Úspěšné nastavení instance Moq objektu

4.2 Řešení pomoci Dependency injection

Korektní řešení je pomocí Dependency injection. Jedná se o techniku pro vkládání závislostí mezi jednotlivými komponentami programu tak, aby jedna komponenta mohla používat druhou.

Následující kód zobrazuje jednoduché rozhraní pro implementaci repositáře.

```
public interface ICompanyRepository
{
    bool InsertCompany(Company company);
}
```

Zdroj. kód 12 - Rozhraní ICompanyRepository

Poté stačí vytvořit instanci manažeru, který pomocí Dependency injection získává repositář, který výše zmíněné rozhraní implementuje.

```
public class CompanyManager
{
    ICompanyRepository company;

    public CompanyManager(ICompanyRepository company)
    {
        this.company = company;
    }

    public bool InsertCompany(Company company)
    {
        return this.company.InsertCompany(company);
    }
}
```

Zdroj. kód 13 - Manažer využívající Dependency Injection

```
[TestMethod]
public void TestInsertCompany()
{
    Company company = new Company() { name = "TCS" };

    Mock<ICompanyRepository> companyRepositoryMock = new Mock<ICompanyRepository>();
    companyRepositoryMock.Setup(x => x.InsertCompany(company)).Returns(true);

    var companyManager = new CompanyManager(companyRepositoryMock.Object);
    Assert.IsTrue(companyManager.InsertCompany(company));
}
```

Zdroj. kód 14 - Předání instance Moqu pomocí Dependency Injection

Zdrojový kód č. 14 zobrazuje, že pomocí techniky Dependency injection není potřeba vytvořit setter pro jednotlivé testované objekty. Tento způsob dodržuje princip testování jednotky.

4.3 Testování datové vrstvy

Jako první byly testovány funkce a třídy, které pracovali s databází. Tedy repositáře. Vždy šlo o to vytvořit Moq objekt repositáře, vložit do Manageru a poté otestovat chování a návratové hodnoty.

```
[TestMethod]
public void TestFindByUrl()
{
    Product product = new Product
    {
        ProductId = 1,
        Url = "url1"
    };

    Mock<IProductRepository> productRepositoryMock = new Mock<IProductRepository>();
    productRepositoryMock.Setup(x => x.FindByUrl("url1")).Returns(product);

    var productManager = new ProductManager();
    productManager.SetRepository(productRepositoryMock.Object);

    Assert.AreEqual(productOne, productManager.FindProductByUrl("url1"));
}
```

Zdroj. kód 15 - Test metody FindByUrl

```
[TestMethod]
public void TestFindByCategoryId()
{
    Product productOne = new Product {ProductId = 1, Url = "url1"};
    Product productTwo = new Product{ProductId = 2, Url = "url2"};
    Product productThree = new Product{ProductId = 3, Url = "url3"};
    var listOfProducts = new List<Product>();
    listOfProducts.Add(productOne);
    listOfProducts.Add(productTwo);
    listOfProducts.Add(productThree);

    Mock<IProductRepository> productRepositoryMock = new Mock<IProductRepository>();
    productRepositoryMock.Setup(x => x.FindByCategoryId(1)).Returns(listOfProducts);

    var productManager = new ProductManager();
    productManager.SetRepository(productRepositoryMock.Object);

    Assert.AreEqual(listOfProducts, productManager.FindByCategoryId(1));
}
```

Zdroj. kód 16 - Test metody FindByCategoryId

```
[TestMethod]
public void TestFindProdutById()
{
    Product productOne = new Product { ProductId = 1, Url = "url1" };
    Mock<IProductRepository> productRepositoryMock = new Mock<IProductRepository>();
    productRepositoryMock.Setup(x => x.FindById(1)).Returns(productOne);

    var productManager = new ProductManager();
    productManager.SetRepository(productRepositoryMock.Object);

    Assert.AreEqual(productOne, productManager.FindProductById(1));
}
```

Zdroj. kód 17 - Test metody FindProductById

4.4 Testování business vrstvy

Při testování této vrstvy se narazilo na první úskalí testování webových aplikací. Došlo k tomu při testování metody GetOrder a CreateOrder, které využívají *cookies* a *session* pro uložení všech položek v košíku. *Cookies* a *session* jsou součástí *HTTP* komunikace a ta nemůže být při testování replikována.

Řešením je obalit tuto práci s *cookies* a *session* do třídy a z té poté vytvořit Moq objekt, který vrátí to, co potřebujeme.

```
public Order GetOrder(int? orderId = null, bool create = true)
{
    if (orderId.HasValue)
        return orderRepository.FindById(orderId.Value);
    if (HttpContext.Current.Session["Order"] != null)
        return (Order)HttpContext.Current.Session["Order"];
    int id = 0;
    string token = null;
    if (BusinessHelper.GetCookie("order") != string.Empty)
    {
        NameValueCollection queryString =
System.Web.HttpUtility.ParseQueryString(BusinessHelper.GetCookie("order"));
        int.TryParse(queryString.Get("order_id"), out id);
        token = queryString.Get("token");
    }
    if (!create && id == 0)
        return null;
    Order order = orderRepository.FindOrderByIdTokenState(id, token,
OrderState.CREATED);
    if (order == null)
        order = CreateOrder();
    HttpContext.Current.Session["Order"] = order;
    return order;
}
```

Zdroj. kód 18 - Práce s cookies a session před úpravou

```
public class BusinessHelper : IBusinessHelper
{

    public void SetCookie(string key, string value, TimeSpan expires)
    {
        HttpCookie encodedCookie = new HttpCookie(key, value);
        if (HttpContext.Current.Request.Cookies[key] != null)
        {
            var cookieOld = HttpContext.Current.Request.Cookies[key];
            cookieOld.Expires = DateTime.Now.Add(expires);
            cookieOld.Value = encodedCookie.Value;
            cookieOld.HttpOnly = true;
            HttpContext.Current.Response.Cookies.Add(cookieOld);
        }
        else
        {
            encodedCookie.Expires = DateTime.Now.Add(expires);
            HttpContext.Current.Response.Cookies.Add(encodedCookie);
        }
    }

    public string GetCookie(string key)
    {
        string value = string.Empty;
        HttpCookie cookie = HttpContext.Current.Request.Cookies[key];
        if (cookie != null)
        {
            HttpCookie decodedCookie = cookie;
            value = decodedCookie.Value;
        }
        return value;
    }

    public bool IsOrderInSession()
    {
        return HttpContext.Current.Session["Order"] != null;
    }

    public Order GetOrderFromSession()
    {
        if (IsOrderInSession())
        {
            return (Order)HttpContext.Current.Session["Order"];
        }
        return null;
    }

    public void SaveOrderIntoSession(Order order)
    {
        HttpContext.Current.Session["Order"] = order;
    }

    public void DeleteOrderFromSession()
    {
        HttpContext.Current.Session["Order"] = null;
    }
}
```

Zdroj. kód 19 - Obalová třída pro práci s cookies a session

Všechna práce s *cookies* a *session* byla přesunuta do třídy BusinessHelper implementující rozhraní IBusinessHelper.

```
public Order GetOrder(int? orderId = null, bool create = true)
{
    if (orderId.HasValue)
        return orderRepository.FindById(orderId.Value);
    if (businessHelper.IsOrderInSession())
        return businessHelper.GetOrderFromSession();
    int id = 0;
    string token = null;
    if (businessHelper.GetCookie("order") != string.Empty)
    {
        NameValueCollection queryString =
HttpUtility.ParseQueryString(businessHelper.GetCookie("order"));
        int.TryParse(queryString.Get("order_id"), out id);
        token = queryString.Get("token");
    }
    if (!create && id == 0)
        return null;
    Order order = orderRepository.FindOrderByIdTokenState(id, token,
OrderState.CREATED);
    if (order == null)
        order = CreateOrder();
    businessHelper.SaveOrderIntoSession(order);
    return order;
}
```

Zdroj. kód 20 - Metoda CreateOrder s využitím BusinessHelperu

Nyní stačí do třídy OrderManager dostat přes setter při testování Moq třídy BusinessHelper a poté lze testovat metody, které *cookies* a *session* používají.

Nejdříve se vytvoří Mock reprezentující BusinessHelper. Poté pomocí metody Setup se provede nastavení, že metoda IsOrderInSession bude vracet true a GetOrderFromSession vrátí objekt typu Order.

```
[TestMethod]
public void TestGetOrderFromSession()
{
    Mock<IBusinessHelper> businessHelperMock = new Mock<IBusinessHelper>();
    businessHelperMock.Setup(x => x.IsOrderInSession()).Returns(true);
    businessHelperMock.Setup(x => x.GetOrderFromSession()).Returns(new Order { });

    var orderManager = new OrderManager();
    orderManager.SetBusinessHelper(businessHelperMock.Object);

    Assert.IsNotNull(orderManager.GetOrder(null, false));
}
```

Zdroj. kód 21 - Získání objednávky ze session při testování

5 ZÁVĚR

E-shop je webová aplikace sloužící ke zprostředkování obchodních transakcí na internetu. Jelikož e-shop není triviální aplikací, zabývá se tato bakalářská práce návrhem a implementací jednotkových testů pro pokrytí funkcionality e-shopu.

Webová aplikace byla vytvořena pomocí frameworku *ASP.NET*, který umožňuje rychle a bezpečně vyvíjet webové aplikace pomocí jazyka *C#*. Pro databázovou vrstvu byl využit *Entity framework*, který poskytuje objektově-relační mapování. Celá aplikace je postavena na architektonického modelu *MVC*.

V textu práce byly důkladně popsány a zobrazeny jednotkové testy. Dále byly popsány ostatní typy testů. Pro testování bylo realizováno 40 jednotkových testů, které pokrývají a testují především business a datovou vrstvu. Pro jednotkové testy byl použit framework *MS Test* a pro tvorbu mock objektů framework *Moq*.

Problém izolace testů byl vyřešen pomocí setterů, kterými se při jednotkovém testování předávaly Mock objekty. Avšak tato technika není zcela správná, neboť při testování by mělo být dodrženo pravidlo, že je možné změnit kód jednotkového testu, ale aplikační kód se nesmí změnit pro test jednotky. Aby toto pravidlo nebylo porušeno, bylo by potřeba využít techniky *Dependency injection*.

Při testování webových aplikací lze narazit na problém při testování funkcí, které používají součásti *HTTP* požadavku, a tudíž při lokálním požadavku nedostupné. V této práci to byly *cookies* a *session*. Řešením je vytvořit pro takové funkce abstrakci v podobě třídy a rozhraní a následně nad touto abstrakcí vytvořit *Mock* objekt.

POUŽITÁ LITERATURA

[1] FREEMAN, Adam. Pro ASP.NET MVC 5. 5th ed. New York, N.Y.: Apress, 2013. ISBN 9781430265306

[2] NAYLOR, Lee. ASP.NET MVC with Entity Framework and CSS. United Kingdom: Apress, 2016. ISBN 9781484221372

[3] MCWHERTER, Jeff a Ben HALL. Testing ASP.NET Web Applications. Indianapolis, Indiana: John Wiley, 2011. ISBN 9781118081228

[4] MVC Recommended Resources. Microsoft Docs [online]. 2014 [cit. 2017-10-19]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/recommended-resources-for-mvc

[5] Entity Framework 6 *Microsoft Docs* [online]. [cit. 2018-04-17]. Dostupné z: https://docs.microsoft.com/cs-cz/ef/ef6/

[6] 6. díl – Scaffolding a Entity Framework v ASP.NET MVC *ITnetwork* [online]. [cit. 2018-04-17]. Dostupné z: https://www.itnetwork.cz/csharp/asp-net/mvc/asp-dot-net-mvc-tutorial-scaffolding-entity-framework-editor-clanku

[7] 1. díl – Popis MVC architektury *ITnetwork* [online]. [cit. 2018-04-17]. Dostupné z: https://www.itnetwork.cz/php/mvc/objektovy-mvc-redakcni-system-v-php-popis-architektury

[8] ASP.NET *The ASP.NET Site* [online]. [cit. 2018-04-17]. Dostupné z: https://www.asp.net/

[9] 1. díl – Úvod do ASP.NET *ITnetwork* [online]. [cit. 2018-04-17]. Dostupné z: https://www.itnetwork.cz/csharp/asp-net/tutorial-uvod-do-asp-dot-net

[10] Intro to Mocking with Moq *Atomic Object* [online]. [cit. 2018-04-17]. Dostupné z: https://spin.atomicobject.com/2017/08/07/intro-mocking-moq/

[11] Jak na akceptační testování pomocí Codeception *Péhápkáři.cz* [online]. [cit. 2018-04-20]. Dostupné z: https://pehapkari.cz/blog/2017/02/09/jak-na-testovani-pomoci-codeception/

[12] Fáze a úrovně provádění testů Testování softwaru [online]. [cit. 2018-04-20]. Dostupné z: http://testovanisoftwaru.cz/tag/akceptacni-testovani/

[13] Unit testing frameworks in C#: Comparing XUnit, NUnit and Visual Studio *Raygun Blog* [online]. [cit. 2018-04-22]. Dostupné z: https://raygun.com/blog/unit-testing-frameworks-c/

[14] Test Doubles – Fakes, Mocks and Stubs *Pragmatists* [online]. [cit. 2018-04-22]. Dostupné z: https://blog.pragmatists.com/test-doubles-fakes-mocks-and-stubs-1a7491dfa3da

[15] Novinky pro testování aplikací *Živě.cz* [online]. [cit. 2018-04-24]. Dostupné z: https://www.zive.cz/clanky/novinky-pro-testovani-aplikaci/sc-3-a-155014/default.aspx

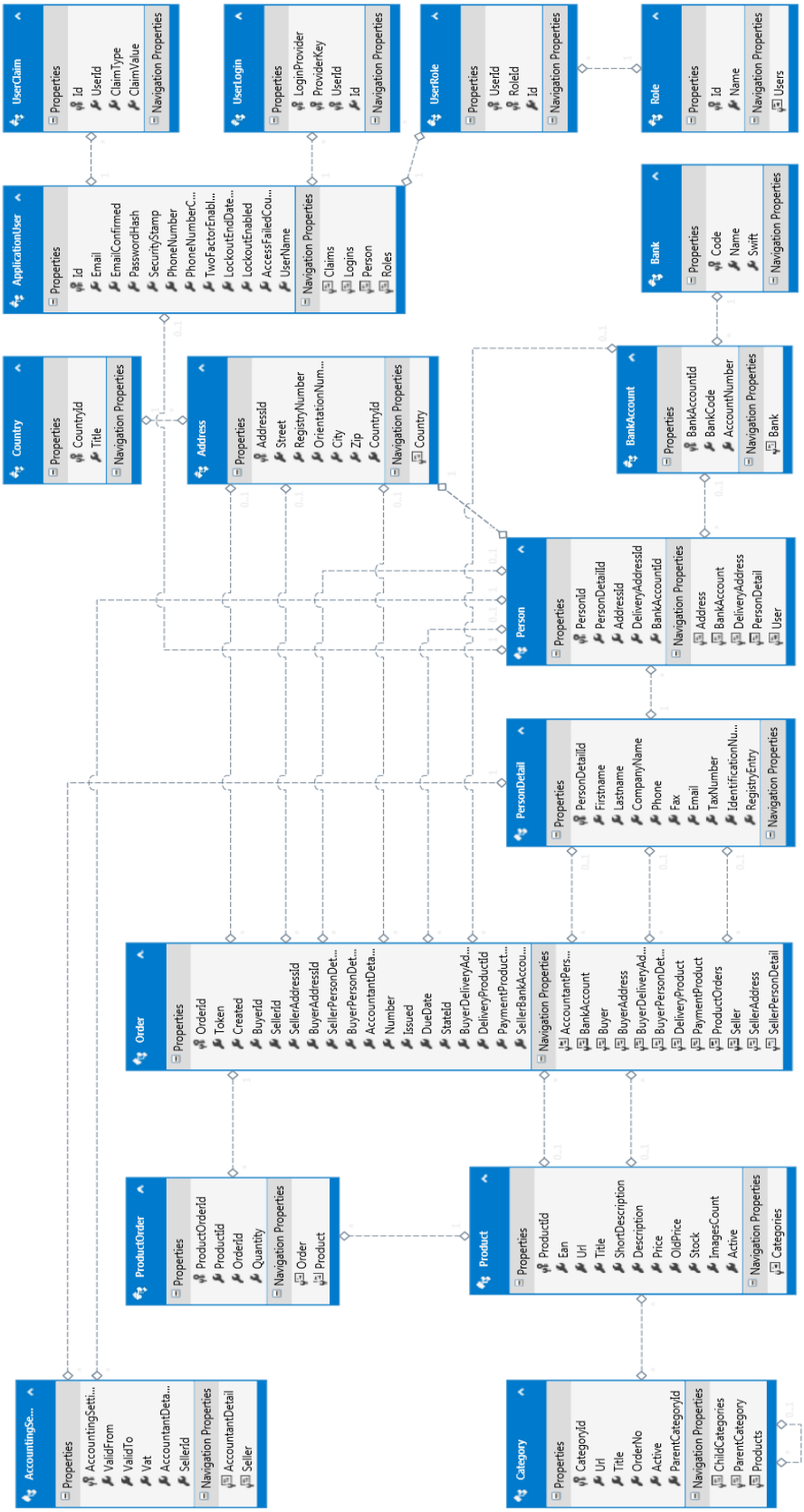
[16] ČÁPKA, David. 1. díl – Popis MVC architektury *ITnetwork* [online]. [cit. 2018-04-24]. Dostupný na WWW: https://itnetwork.cz/images/5/patterns/mvc.png

[17] Lipski, Michal. Test Doubles – Fakes, Mocks and Stubs *Pragmatists* [online]. [cit. 2018-04-24]. Dostupný na WWW: https://cdn-images-1.medium.com/max/800/0*snrzYwepyaPu3uC9.png

[18] Lipski, Michal. Test Doubles – Fakes, Mocks and Stubs *Pragmatists* [online]. [cit. 2018-04-24]. Dostupný na WWW: https://cdn-images-1.medium.com/max/800/0*KdpZaEVy6GNnrUpB.png

[19] Lipski, Michal. Test Doubles – Fakes, Mocks and Stubs *Pragmatists* [online]. [cit. 2018-04-24]. Dostupný na WWW: https://cdn-images-1.medium.com/max/800/0*k7mwTF60slyMxRlm.png

SEZNAM PŘÍLOH

Příloha A – ER diagram databáze 39

PŘÍLOHA A – ER DIAGRAM DATABÁZE