Webová aplikace pro kompletní správu automatizovaných testů v agilním prostředí

Bc. Ondřej Jakuba

Diplomová práce
2019

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2018/2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Ondřej Jakuba**
Osobní číslo:	**A16197**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**kombinovaná**
Téma práce:	**Webová aplikace pro kompletní správu automatizovaných testů v agilnim prostředí**
Téma anglicky:	**A Web Application for the End-to-End Management of Automation Testing in Agile Environments**

Zásady pro vypracování:

1. **Nastudujte danou problematiku související s automatizovanými testy a frameworky.**
2. **Vyberte vhodné metody a nástroje pro realizaci webové aplikace pro spouštění automatizovaných testů.**
3. **Vytvořte design a architekturu webové aplikace.**
4. **Implementujte webovou aplikaci za využití vámi vybraných metod a nástrojů.**
5. **Proveďte test aplikace a vhodně reprezentujte získané výsledky.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **SPAANJAARS, Imar. Beginning ASP.NET 4.5 in C# and VB. Indianapolis, IN: Wiley, c2013. Wrox beginning guides. ISBN 9781118311806**
2. **ŠOCHOVÁ, Zuzana a Eduard KUNCE. Agilní metody řízení projektů. 1. Environment. Brno: Computer Press, 2014, 175 s. ISBN 978-80-251-4194-6.**
3. **MYSLÍN, Josef. Scrum: průvodce agilním vývojem softwaru. 1. vydání. Brno: Computer Press, 2016, 167 s. ISBN 978-80-251-4650-7.**
4. **HUGGINS, Jason, Gheorghe Gheorghiu, C. Titus BROWN, An Introduction to Testing Web Applications with twill and Selenium**
5. **LUIS, Feroz Pearl, Gaurav Gupta, Mastering Mobile Test Automation**

Vedoucí diplomové práce: **Ing. Petr Žáček**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **3. prosince 2018**

Termín odevzdání diplomové práce: **15. května 2019**

Ve Zlíně dne 7. prosince 2018

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
garant oboru

ABSTRAKT

Testování software neodmyslitelně patří k jeho samotnému vývoji. Automatizace testů v agilním vývoji velmi často šetří prostředky, nicméně někteří testeři tuto automatizaci nemusí zvládat snadno. Zvládají však většinou napsat vhodný testovací scénář. Tato problematika mě vedla k vytvoření webové aplikace, která těmto testerům usnadní práci. Nalezl jsem několik nástrojů, které mají za cíl téměř totožné, avšak nesplňovalo to zcela mou představu.

Pro implementaci webové aplikace jsem využil ASP.NET MVC 5. Testovací aplikaci jsem napsal v .NET Framework 4.5, ve které jsem použil nástroj Selenium.

Svou aplikaci jsem však dokončil jen částečně. Na použití to však vliv nemá. Aplikace je funkční a tvoření testů v této aplikaci působí přívětivě a poměrně přehledně. První nedostatky se také objevily. Při dalších úpravách bych se soustředil na vhodnější dohledání elementů a na možnost vytvářet skupiny kroků, které lze použít ve více testech. To by výrazně prospělo ve větších projektech.

Klíčová slova:

Agilní vývoj, Testování, Selenium, ASP.NET, MVC, C#

ABSTRACT

Software testing is inherent to software development. Test automation saves resources in agile development very often. However some testers are not able to create automation testing. They can usually write a suitable test scenario. This issue has let me to create a web application that can be used by these testers for creating automation tests. I have found several tools that aim to be almost identical. These tools do not fulfill my idea completely.

I used ASP.NET MVC 5 to implement the web application. I wrote the test application in .NET Framework 4.5 and I used the Selenium tool.

I completed my application only partially. However it has no effect to use. The application is functional and the test creation is friendly and fairly clear. The first deficiencies

also appeared. My next edits will focus on finding the elements more appropriate and on creating groups of steps that can be used in multiple tests. This would benefit greatly in larger projects.

Keywords:

Agile development, Testing, Selenium, ASP.NET, MVC, C#

Nejprve bych chtěl poděkovat svému vedoucímu práce, kterým je Ing. Petr Žáček, za to, že mi umožnil pracovat na této diplomové práci. Dále bych chtěl poděkovat své rodině, která mi byla oporou po celé studium. Poděkování si zaslouží také BcA. Lívia Jančíková za pomoc s vytvořením ikonek, které jsem v této práci použil a také za to, že při mně celé studium stála. Další poděkování patří firmě CN Group CZ s.r.o., kde jsem získal spoustu cenných zkušeností a znalostí na pozici testera a umožnila mi získat certifikaci ISTQB.

Přečtením mé diplomové práce získáte několik informací o testování software. Dovíte se, jak lze přistupovat k vývoji SW a také budete znát různé druhy testování, které se v praxi běžně používají. Téma testování jsem si vybral, protože jsem byl 3 roky zaměstnán jako inženýr testování software a získal jsem certifikáty ISTQB CTFL a ISTQB Agile Tester. V teorii vám předám informace, o kterých by měl mít přehled každý tester a v praxi vás seznámím s mým návrhem a provedením aplikace, ve které jsem si dal za cíl usnadnit práci testerům, při tvorbě automatizovaných testů.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 17.5.2019

…….
podpis diplomanta

OBSAH

ÚVOD

V této diplomové práci rozebírám problematiku testování v agilním prostředí. Jde o téma v dnešní době velmi důležité, neboť mnoho firem pochopilo, že kvalita software je velice důležitá a dodávání kvalitního software zajišťuje vývojové firmě důležitou prestiž. Tomu odpovídá i trh práce, kde poptávka po testerech, kteří mají odpovídající profil je stále vysoká. Mě samotného oslovují několikrát za měsíc „lovci zaměstnanců“ s nabídkou práce v tomto směru. A co víc? Dnešní doba přímo vyžaduje nespočet aplikací. Dostupnost k informačním technologiím je dnes natolik vysoká, že i má babička je schopná vyhledávat informace na internetu. Lidé stále více využívají webové služby k nákupům a jiným činnostem, neboť v tom sami pociťují jistou pohodlnost. Tato skutečnost tak tlačí především firmy, které chtějí být úspěšné k tomu, aby si nechali vytvořit také webové aplikace. Z tohoto důvodu se tedy v oboru IT protočí poměrně vysoká část financí.

Během své pracovní činnosti testera, kde jsem se pohyboval mezi kolegy na stejné pozici, jsem zjistil, že mají někteří kolegové problém s psaním kódu, což je omezovalo na manuální testování. Někteří možná jen nedostali příležitost, jiní měli problém s pochopením pro kódování a je pro ně náročné psát vlastní kusy kódu. Na základě tohoto problému mě napadlo vytvořit něco, co by jim ve své práci pomohlo a bylo tak užitečné.

Někdo může namítnout, že již existuje nespočet možností na internetu a že vlastně se snažím o něco, co už existuje. Já bych však rád předčil tyto nástroje a poskytl vhodnější řešení. Mnoho dostupných nástrojů je především finančně nedostupné pro menší firmy. Některé nástroje mají své nedostatky, které brání uživateli být více produktivní a tak dále. Nemyslím si, že by tyto nástroje byly špatně navržené, nebo vytvořené, jen existuje několik způsobů přístupu, jakým lze testy vytvářet. Odborníci na téma tvorby testů ve svých pracích představují kvalitní doporučení, kterých se držet při tvorbě testovacích scénářů i ohledně výběru vhodného nástroje pro samotnou automatizaci. Je tedy třeba myslet na to, že každý projekt, na kterém může tester spolupracovat, je svým způsobem jedinečný a je třeba uvážit, zda nástroj, který se osvědčil na jednom projektu, bude vhodný i pro projekt další. Nejlepší volba je tedy velice individuální.

V první části popisuji druhy přístupů k vývoji software. U těchto modelů popisuji výhody a hlavní myšlenku. Nejvíce se pak rozepisuji u modelu pro agilní vývoj, neboť k tomuto modelu míří má diplomová práce.

V druhé části pojednávám o testování software. Vysvětluji zde, jaké přístupy k testování jsou používány. U testovacích úrovní popisuji jejich význam a použití. Snažím se zde poukázat i na chyby, které vznikají v této souvislosti. Také zde uvádím, kdo testování v jednotlivých úrovních provádí. Poté uvádím podmínky, kdy použít manuální a automatizované testování.

Ve třetí části jsem popsal testování v agilním prostředí. Konkrétně tvorbu uživatelských příběhů a důležitou roli testera při jejich tvorbě, důvody k pravidelné retrospektivě a také výhody použití automatizovaných testů pro continuous integration.

Ve čtvrté části se zabývám nástroji pro automatizaci testování. Zde popisuji důvody pro použití frameworků, jaké frameworky existují a k čemu slouží. Také zde uvádím některé existující nástroje, které lze použít k vytvoření automatizovaných testů.

V páté části se věnuji výběru nástrojů a metod. Především pak odůvodnění mého výběru technologie a popisem, jak pracuje.

V šesté části popisuji vlastní návrh aplikace, na které jsem pracoval. Nastínil jsem zde aspekty, se kterými jsem pracoval a vysvětlil jsem mé důvody k vytvoření svého návrhu, tak aby návrh odpovídal skutečným požadavkům. Jde především o prvky, které je nutné použít a které popisují data, která bude aplikace shromažďovat. Také pak návrh designu.

V sedmé části se již zabývám samotnou implementací aplikace. Popisuji důležité kroky, které jsou nezbytné pro vytvoření aplikace způsobem, který jsem si zvolil. Pomocí diagramů se snažím popsat architekturu aplikace. Mimo to zde také poukazuji na způsob tvorby aplikace a kroky, které vedly k úspěšnému dokončení aplikace.

Osmá část je již ukázkou výsledné aplikace. Popisuji zde postup pro práci s aplikací. V této části jsem vytvořil názorný test, který jsem také spustil a ověřil tak, že mé řešení je funkční.

V rámci této diplomové práce řeším tyto dílčí úkoly:

- Nastudujte danou problematiku související s automatizovanými testy a frameworky
- Vyberte vhodné metody a nástroje pro realizaci webové aplikace pro spouštění automatizovaných testů
- Vytvořte design a architekturu webové aplikace

- Implementujte webovou aplikaci za využití vámi vybraných metod a nástrojů
- Proveďte test aplikace a vhodně reprezentujte získané výsledky.

I. TEORETICKÁ ČÁST

1 VÝVOJ SOFTWARE

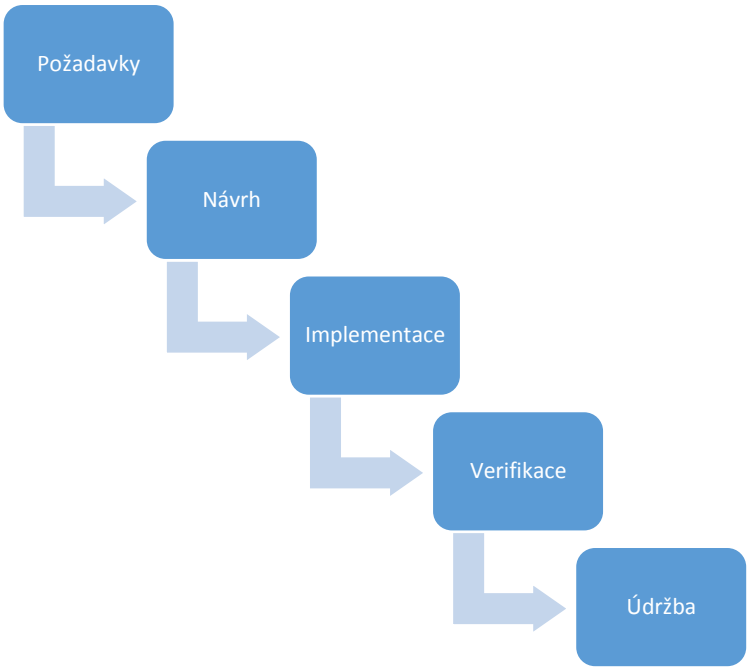
Ve vývoji software existuje spousta doporučení a pohledů na věc. Než člověk začne se samotným testováním software, měl by mít alespoň obecný přehled, jakým způsobem samotný software vzniká a následně pak se zabývat otázkou, proč vlastně software testujeme. Před samotným vývojem je důležité vytvořit dobrý návrh. Především pak, pokud vyvíjíme pro zákazníka, je potřeba důkladně prokonzultovat veškeré požadavky. Na základě požadavků je vhodné vytvoření modelu. K modelování můžou vhodně posloužit nástroje, jako je jazyk UML, CASE nástroje, ale i jiné. Při modelování jsou navrhovány k přehlednému zobrazení jednotlivých dílčích částí SW, k návrhu vztahů a spolupráce mezi nimi. Také k návrhu vstupů a výstupů jednotlivých částí. Modelování má pak dopad na samotný průběh vývoje. Vývoj je více organizovaný a předchází před tvorbou nedomyšlených kusů kódů.

1.1 Přístup k vývoji SW

K vývoji SW lze přistupovat několika způsoby. Jsou dva základní typy přístupů a to sekvenční (lineární) a iterativní. Hlavním modelem pro sekvenční přístup je model vodopádový. Zástupcem iterativního přístupu je prototypový model. Kombinací lineárního a iterativního přístupu pak vznikají modely jako inkrementální a spirálový.

1.1.1 Vodopádový model

Hlavní myšlenkou je postup jedním směrem, z jednoho stádia do druhého bez možnosti cesty zpět, podobně jako tomu je u vodopádů, kde voda teče jen směrem dolů.

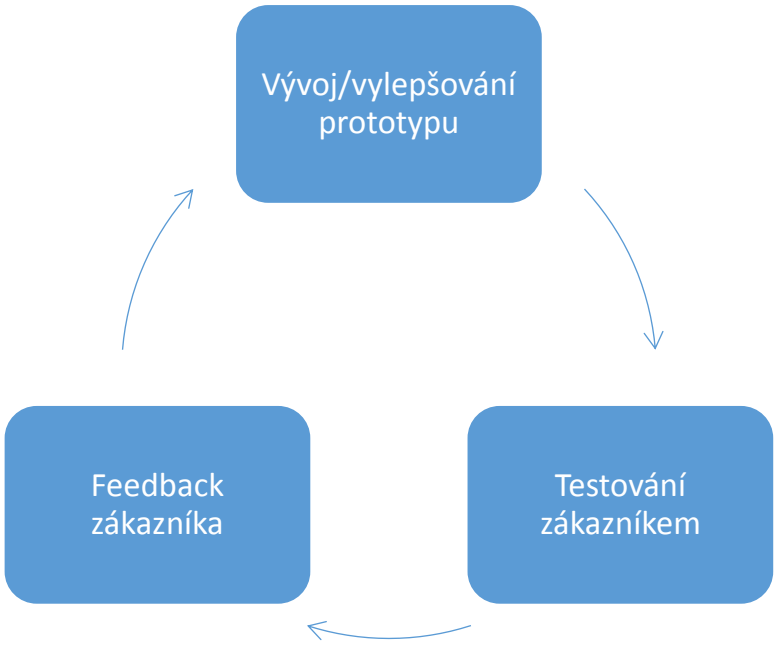


Obrázek 1: Vodopádový model

V tomto modelu je důležité plánování, přesné časové rozvržení a rozpočty. Celý vývoj SW se realizuje najednou.

1.1.2 Prototypový model

Jde o způsob vývoje SW, kde jsou vyvíjeny prototypy SW, který je testován zákazníkem a na základě zpětné vazby zákazníka je SW dále vyvíjen.

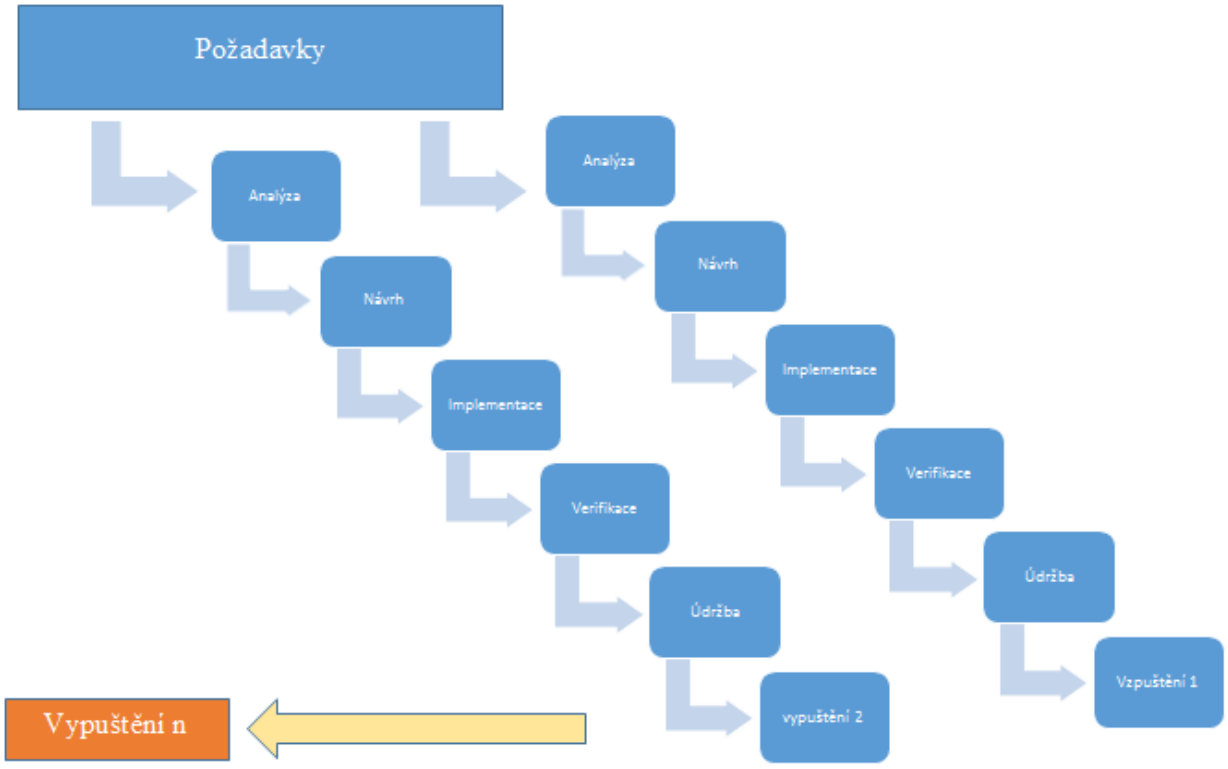


Obrázek 2: Prototypový model

Snahou tohoto modelu je snížení nebezpečí projektových rizik. Rozdělením vývoje na menší celky lze snadněji dosáhnout změny v průběhu vývoje. Další výhodou je, že je zákazník součástí vývoje a tak je větší pravděpodobnost uspokojení zákazníka. (1)

1.1.3 Inkrementální model

Inkrementální, tedy přírůstkový model funguje na principu nabalování přírůstků. Každý přírůstek je vyvíjen např. po vzoru modelu typu vodopád.

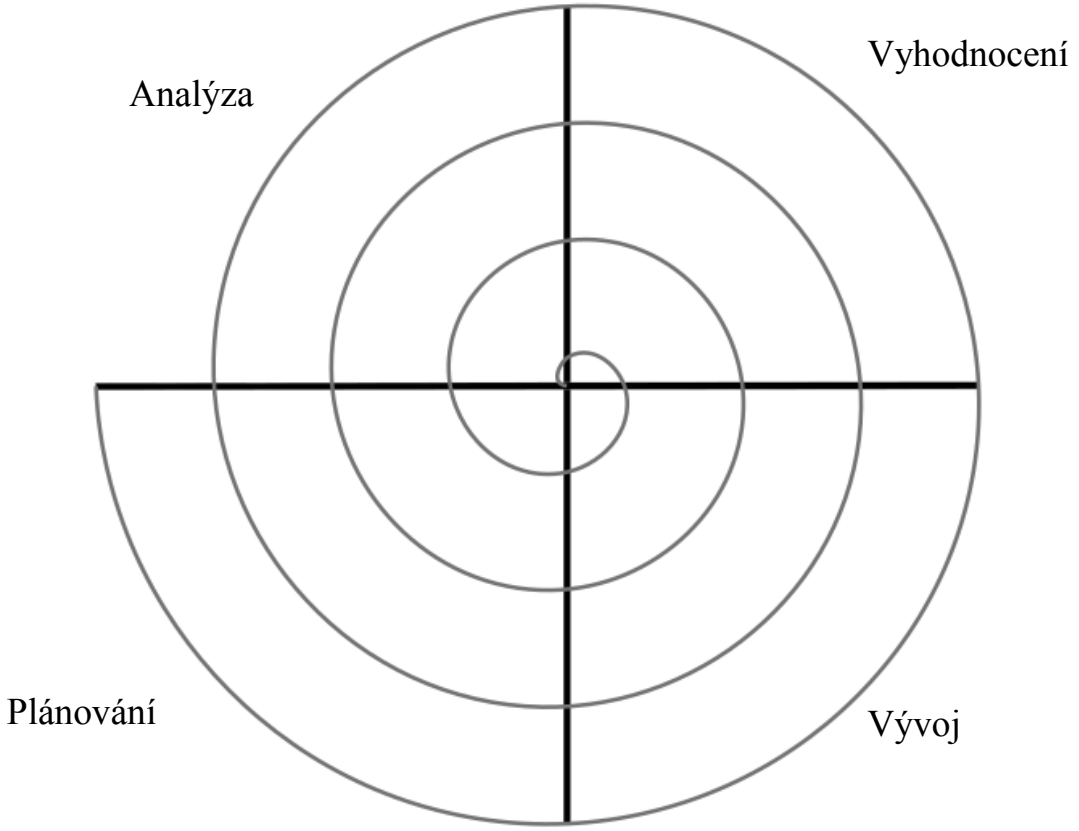


Obrázek 3: Inkrementální model

Cílem je omezit rizika selhání produktu rozdělením projektu na menší části. Výhodou může být to, že si požadavky rozebere více týmů a pracují na nich souběžně a přitom nezávisle. (2) (3)

1.1.4 Spirálový model

Spirálový model se prezentuje graficky jako spirála, u které se smyčka nazývá fáze vývoje. Počet těchto smyček se mění od projektu k projektu. Poloměr spirály značí vynaložené náklady na vývoj, zatím co úhlový rozměr odpovídá dosavadnímu pokroku. Každá fáze je rozdělena do čtyř kvadrantů (Analýza, vyhodnocení, vývoj a plánování). (4) (5)



Obrázek 4: Spirálový model

1.1.5 Rapid Application Development (RAD)

Důraz na rychlost a nízkou cenu vývoje. Jde-li produkt rozdělit na malé moduly, lze implementovat tento model tak, že jsou přiřazeny jednotlivé modely jednotlivým týmům, které na sebe nejsou závislé. Vývoj každého modulu pak zahrnuje kroky jako u vodopádového modelu. (6) (5)

1.1.6 Agilní vývoj

Agilní vývoj se soustředí na usnadnění rychlého dokončení projektu. V agilním vývoji se neplýtvá časem a energií na činnosti, které nejsou nezbytné. Požadavky jsou rozděleny na menší části. Ty jsou pak vyvíjeny postupně. Agilní vývoj se zakládá na iterativním přístupu. Z pravidla jsou iterace malé a snadno docílitelné a to během několika dnů až týdnů. V každé iteraci se provádí plánování, vývoj a doručení zákazníkovi. V krocích tedy shromáždění požadavků, následně jejich analýza, návrh, samotné kódování, kde následují jednotkové testy a nakonec akceptační testy.

V agilním vývoji je nejdůležitější uspokojit zákazníka. To se snaží docílit rychlým a průběžným dodáváním kvalitního SW. Obrovskou výhodou může být to, že v agilním vývoji není tolik kladen důraz na dokumentaci, pokut nějaká dokumentace vůbec vzniká. Namísto dokumentace je důležitější fungující SW. Průběžným doručováním SW zákazníkovi lze včas reagovat na změnu. V agilním vývoji se počítá s tím, že zákazník většinou sám neví, co vlastně chce. Požadavky na změny jsou tedy vítány i v průběhu vývoje. Dalším parametrem spojeným se zmíněným je i to, že spolupráce mezi lidmi je důležitější než nástroje a procesy. Tato spolupráce probíhá skrz na skrz všemi podílejícími se na projektu. Tedy i lidi z businessu spolupracují s developery a to na každodenní bázi během celého vývoje. Důležité je mít motivované jedince. Doporučuje se, aby tým byl malý (5-9) osob. V malém týmu je větší potenciál, že se do komunikace zapojí všichni. Existuje zde idea, že informovanost celého týmu dosáhneme efektivněji osobním kontaktem, než předáváním formálních dokumentů.

Agilní vývoj zavádí i párové programování. To je založeno na ideji, že párové programování vyprodukuje kvalitnější program s méně chybami, než kdyby totéž měl programovat jednotlivec. Párové programování je založeno na střídání rolí dvou programátorů, kdy jeden píše kód a druhý dělá revizi.

Ve výsledku lze tedy říci, že jde o vývoj, který snižuje celkovou dobu vývoje a při tom má zástupce zákazníka přehled po celý vývoj o aktuálním stavu produktu, díky čemuž může kdykoliv pozměnit požadavky. Nevýhodou však může být absence formálních dokumentů a tak může docházet ke špatné interpretaci mezi členy týmu a po dokončení produktu, kdy vývojáři již pak jsou přeřazeni na jiný projekt, může být problematická údržba.

Nejznámějšími zástupci agilního modelu jsou SCRUM, Extreme programming (XP) a test driven development (vývoj řízen testy). (7) (8) (9)

2 TESTOVÁNÍ

Testování software je nedílnou součástí vývoje. Testováním lze dosáhnout zvýšení kvality. Testovat SW lze několika různými způsoby, které lze také kombinovat. Některé testy se zaměřují na bezpečnost programu, jiné na výkon a pak především na samotnou funkčnost. Způsob testování lze rozdělit několika způsoby.

2.1 White/black box

Základním rozdělením testování pohled na program, kde buď máme možnost vidět jen výsledný produkt z pohledu uživatele, nebo máme možnost číst kód samotného programu.

Black box testing – jedná se o způsob testování, kdy tester nezná samotnou strukturu programu a při testování vychází jen z požadavků a popisu o tom, jak by se měl výsledný produkt chovat. Při testování se tedy vychází z chování programu a to tak, že sledujeme jaký výsledek dostaneme po zadání vstupních parametrů. (10)

White box testing – v tomto případě má tester k dispozici zdrojový kód a má znalosti struktury programu. Díky tomu má tester více možnosti k otestování všech možných průchodů kódem a k otestování neočekávaných vstupních hodnot. (11)

Gray box testing – je založen na kombinaci obou předchozích pohledů. (12)

2.2 Úrovně testování

Testovací úrovně souvisí s životním cyklem vývoje. V sekvenčních modelech životního cyklu jsou úrovně definovány tak, že výstupní kritéria jedné úrovně jsou součástí vstupních kritérií pro další úroveň. Toto pravidlo ale neplatí pro některé iterativní modely. Během iterací každá daná user story bude typicky probíhat sekvenčně skrze následující aktivity:

- Unit testy, které jsou obvykle prováděny vývojáři
- Akceptační testy, které mohou být rozděleny do následujících dvou
    - Testy ověřující funkčnost, které jsou často automatizované, mohou být prováděny jak vývojáři, tak testery
    - Testy ověřující funkčnost, které jsou obvykle manuální. Můžou být prováděny vývojáři, testery, ale také obchodními partnery spolupracujícími na ur-

čení, zda je funkce vhodná k použití, ke zpřesnění dosaženého pokroku a aby poskytli zpětnou vazbu.

Současně se navíc provádí často i regresní testování, které zahrnuje opakované spouštění automatizovaných jednotkových testů a verifikační testy funkcí jak z aktuální iterace, tak i z iterací minulých. (13) (14) (15)

2.2.1 UNIT testy

Zaměřují se na testování komponent (nejmenší testovatelná část SW). Klíčovou roli hrají při ujištění se v projektech, kde se průběžně mění kód, že tyto změny neporušili stávající komponenty. UNIT testy mohou zamezit uniknutí chyb do vyšších testovacích vrstev. Nalezené chyby v této vrstvě jsou běžně opraveny, jakmile jsou nalezeny, bez vytváření oficiálních reportů. Oprava v této úrovni je v podstatě nejméně nákladná, jelikož k nalezení chyby stačí projít poslední napsané úpravy. Testy jsou psány obvykle vývojáři současně s psaním testovaného kódu. (16) (14)

2.2.2 Integrační testy

Integrační testy se zaměřují na testování závislostí mezi komponenty a systémy. Stejně jako u jednotkových testů, integrační testy ověřují, zda nedošlo k poškození stávajících rozhraní, komponent nebo systémů. Pokud moduly tvoří nedílný celek, testování by mělo být zaměřeno na testování komunikace, mezi jednotlivými moduly, ne na testování modulů samotných. V případě, že tvoří nedílný celek systémy, pak se zaměřuje na komunikaci mezi systémy. Samotnou funkčnost systémů testujeme v systémovém testování.

Integraci komponent testují většinou vývojáři, zatím co integraci systémů testují testeři. (14)

2.2.3 Systémové testování

Systémové testování se zaměřuje na celkové chování a vlastnosti systému jako celek. Tato úroveň testování také přináší jisté ověření, zda s novými změnami nedošlo k poškození existující vlastnosti. Jde o testování produktu z pohledu zákazníka. Provádí se před doručením k zákazníkovi. Výsledky z testování mohou pomoci rozhodnout zájmovým stranám o vydání SW. Testovací prostředí by mělo ideálně odpovídat tomu výslednému, či produkčnímu. Testování probíhá na základě testovacích scénářů, které popisují scénář,

který by mohl nastat v praxi. Po případném nalezení a opravení chyby je třeba tyto testy opakovat. Testování provádí obvykle testeři, kteří jsou nezávislí. Pokud specifikace nejsou dostatečné, či chybné, tester může očekávat jiné chování, než je vyžadováno. Tato skutečnost pak může způsobit situaci, že vzniknou plané výsledky (chybové i pozitivní). (17) (14)

2.2.4 Akceptační testování

Akceptační testy se zaměřují na chování a vlastnosti celého produktu, stejně jako u systémového testování. Nalezení chyb však není cílem. Akceptační testování mohou posloužit k posouzení o připravenosti produktu k nasazení a použití zákazníkem.

2.2.4.1 Uživatelské akceptační testování

Zaměřeno na ověření vhodnosti pro použití systému plánovanými uživateli v reálném nebo simulovaném provozním prostředí. Hlavním cílem je ověření, že systém plní potřeby uživatelů, plní požadavky a obchodní procesy s minimálními obtížemi, náklady a riziky.

2.2.4.2 Funkční akceptační testování

Akceptační testování je prováděno obsluhou nebo zaměstnanými správci v produkčním prostředí. Zaměřuje se na provozní aspekty, jako může být instalace, upgrade, zálohování, kontrola bezpečnosti a jiné. Hlavním cílem je ověření, že provozovatel nebo správce může udržovat uživatelům systém v pořádku v provozním prostředí.

2.2.4.3 Smluvní a regulační akceptační testy

Smluvní akceptační testy jsou prováděny vzhledem ke smluvním akceptačním kritériím pro vytvoření SW na zakázku. Tato akceptační kritéria by měla být vytvořena při uzavření spolupráce. Smluvní akceptační testy jsou často prováděny uživateli nebo nezávislými testery.

Regulační akceptační testy jsou prováděny vzhledem ke všem regulacím, které musí být dodrženy. Jsou to třeba vládní, právní nebo bezpečnostní předpisy. Jsou často prováděny uživateli, nebo nezávislými testery někdy s výsledky, které se mohou stát svědectvím nebo kontrolou pro regulační orgány.

2.2.4.4 Alfa a Beta testování

Typicky je používáno komerčními vývojáři, kteří mají zájem získat zpětnou vazbu od potenciálních, nebo stávajících uživatelů před uvedením produktu na trh.

Alfa testování je prováděno na stránkách vývojové organizace, zatím co Beta testování je prováděno na prostření uživatelů. Beta testování může přijít po Alfa testování, ale i bez něj.

Hlavním důvodem je ověřit, že lze systém používat za běžných podmínek a případné zjištění chyb, které jsou závislé na prostředí a situacích, za kterých je produkt užíván.

2.3 Manuální testování

Tester by měl představovat koncového uživatele a zkoušet všechny možné funkce fungují dle specifikací. Během testování tester provádí test case (testovací scénáře) a na základě výsledku produkuje reporty manuálně bez užití některých z automatizačních nástrojů.

2.3.1 Exploratory testing

Manuální testování se využívá především při exploratory (ověřovacích) testech. Jedná se o testování založené na zkušenostech testera. Je tedy vhodné, aby bylo prováděno zkušenými odborníky. Testování se provádí bez znalostí požadavků. Zkoumány jsou funkce aplikace.

2.3.2 Usability testing

Testování použitelnosti slouží k ověření, že je aplikace uživatelsky příjemná. Cílem je vyhodnotit, zda koncový uživatel může snadno pochopit prostředí a bude schopen aplikaci používat.

2.3.3 Ad-hoc testing

Neformální způsob testování, kdy testeři náhodně testují aplikaci bez dodržování dokumentace a bez testovacích návrhů. Primárně je testování prováděno testery se znalostí aplikace.

2.4 Automatizované testování

Testování je prováděno za pomocí automatizačních nástrojů. Jsou spouštěny testovací skripty a výsledky jsou tvořeny automaticky automatizačními nástroji. Ne vždy je však automatizace testů vhodná. V zásadě je důležité, aby testy byly opakovatelné a to i výhledově do budoucna. Dále by měly testovat funkcionality, které jsou již dokončeny.

2.4.1 Regresní testování

Regresní testování je využíváno u agilního vývoje. Opakovaným spouštěním testů lze ověřit, zda postupným vývojem a úpravou programu nedošlo k poškození některé z části programu. Díky častému provádění změn v kódu a tedy častému spouštění totožného testu je regresní testování vhodným příkladem využití automatizovaného testování. Provádění regresních testů manuálně by bylo velmi časově náročné. (18)

2.4.2 Zátěžové testy

Klade si za úkol ověřit produkt, že zvládne očekávaný počet transakcí a normálně fungovat i v období, kdy dochází ke špičce v provozu. Automatizované testování je nejlepším způsobem, jak zátěžové testy provádět. (19)

2.4.3 Testování výkonu

Tento typ testování zjišťuje nebo ověřuje rychlost, škálovatelnost a stabilitu produktu. (19)

3 TESTOVÁNÍ V AGILNÍM PROSTŘEDÍ

3.1 Spolupráce na tvorbě uživatelských scénářů

Uživatelské scénáře (user stories) jsou v agilním vývoji psány k zachycení požadavků z perspektivy vývojářů, testerů a obchodních zástupců. Uživatelské scénáře musí obsahovat funkcionální i nefunkcionální charakteristiky. Každý příběh zahrnuje akceptační kritéria pro tyto charakteristiky. Tato kritéria by měla být definována ve spolupráci mezi vývojáři, testery a obchodními zástupci. V agilním týmu se považuje úkol za dokončený, když jsou akceptační kritéria splněna. Pohled z perspektivy testera může vylepšit uživatelské příběhy tím, že najde chybějící detaily nebo nefunkcionální požadavky. Tester tedy může položit doplňující otázky obchodním zástupcům a navrhne jakým způsobem otestovat tyto uživatelské příběhy a ověří akceptační kritéria.

3.2 Retrospektiva

Jde o setkání, které se koná na konci každé iterace. Cílem je diskutovat o tom, co bylo úspěšné, co by se dalo zlepšit a jak tato zlepšení začlenit a jak pokračovat s úspěchy v následujících iteracích. Výsledkem souvisejícím s testováním jsou rozhodnutí o zaměření se na účinnost testů, produktivitu testů, kvalitu test case a spokojenost týmu. Taktéž se řeší testovatelnost aplikací, uživatelských příběhů, funkcí, nebo systémového rozhraní. Analýza problémů může vylepšit řízení testu a vylepšit vývoj.

3.3 Continuous Integration

Na konci každého sprintu je od rozšíření produktu vyžadována spolehlivost, funkčnost a sjednocení software. Průběžná integrace tuto problematiku řeší průběžným slučováním všech vytvořených změn v SW, a sjednocuje všechny změněné komponenty průběžně alespoň jednou denně. Tím, že vývojáři neustále integrují své změny, mohou být průběžně sestaveny (vytvořen build) a následně otestovány. Díky tomu případné chyby v kódu jsou nalezeny daleko rychleji.

Nicméně kompilace a vydání nové verze je složitý proces. Z tohoto důvodu je vhodné využít nástroj k automatizaci buildu.

Průběžná integrace umožňuje agilním testerům spouštět automatizované testy pravidelně. V některých případech mohou být testy spouštěny v rámci samotného procesu prů-

běžného integrování. To poskytuje rychlou zpětnou vazbu o kvalitě kódu. Tyto výsledky mají možnost vidět všichni členové týmu. Tím, že jsou regresní testy automatizované, testeři se mohou věnovat manuálnímu testování týkajících se nových funkcí a implementačních změn.

4 NÁSTROJE PRO AUTOMATIZACI TESTOVÁNÍ UŽIVATELSKÉHO ROZHRANÍ

Pro ulehčení práce při vytváření testů lze využít nástroje k tomu určené. Těchto nástrojů existuje mnoho. Důležité je vybrat si ten správný.

4.1 Test Automation Framework

Obecně Framework slouží jako podpora při programování. Definuje sadu pravidel a nejlepších postupů, kterými se můžeme řídit pro dosažení požadovaných výsledků. Framework může obsahovat knihovny API, podporu pro návrhové vzory, podpůrné programy a doporučené postupy.

Framework pro automatizované testování je tedy v podstatě soubor pokynů sloužících k vytváření a navrhování testů. Použitím frameworku si tester může ulehčit spoustu práce. Odpadnou tak náležitosti spojené s vytvářením kódu, což vyžaduje jistý stupeň zkušeností a také, o který je třeba se starat. Tester se tak může soustředit na práci spojenou se samotným testováním. (20) (21) (22)

4.1.1 Linearní Skriptovací Framework

Je to základní úroveň automatizačních frameworků, který je ve formě „Record and Playback“ lineárním způsobem. Tento typ je vhodný pro malé aplikace. Vytváření a spouštění testů se provádí pro každý test samostatně.

Výhodou je, že není třeba ztrácet mnoho času tvorbou testovacích skript. Nicméně má nedostatky jako je opakovatelnost, nebo natvrdo zapsaná data, takže není možné spouštět test na různých sadách dat. (23) (22)

4.1.2 Modální testovací Framework

Zde testeři vytváří testovací skripty vhodným rozdělením celé aplikace na menší nezávislé testy. Testeři si tedy rozčlení aplikaci na menší moduly a testovací skripty vytváří individuálně. Z takto vytvořených skriptů lze jejich kombinací vytvářet větší testovací skripty za pomocí hlavního skriptu tak, aby byly pokryty požadované testovací scénáře. Testy se pak spouští za užití vytvořených hlavních testovacích skript. (23) (22)

4.1.3 Data-driven Framework

Daty řízený framework pro automatizaci testů se zaměřuje na rozdělení logiky testovacích skriptů a testovacích dat od sebe. Umožňuje to vytvářet skripty pro automatizované testy předáváním různých sad testovacích dat. Tyto sady testovacích dat jsou uchovávány v externích souborech, nebo ve zdrojích jako je XML soubor, Databáze, sešit Excelu a jiné. Tento Framework výrazně snižuje počet vytvořených skript.na rozdíl od modálního testovacího Frameworku. (23) (22)

4.1.4 Framework pro testování klíčovými slovy

Využívá se tabulka k určení klíčových slov, slov popisujících akci pro každou funkci nebo metodu, kterou bychom mohli provádět. Pomocí tohoto Frameworku můžou testeři vytvářet jakýkoliv testovací skript pro automatizaci užitím těchto klíčových slov. Testeři, kteří mají menší znalosti v programování, jsou tak schopni také vytvářet testovací skripty. Nicméně počáteční práce a nastavení vyžaduje více odbornosti. (23) (22)

4.1.5 Hybrid Driven Testing Framework

Jde o kombinaci dvou a více uvedených frameworků. Snaží se využít více druhů výhod. (23) (22)

4.1.6 Framework pro chováním řízený vývoj testování

Behavior Driven Testing (BDT) velmi uznávaná a běžná metodika v agilním prostředí. Testy se více zaměřují na chování uživatelů, než na technické funkce SW. BDT je nejlepší volbou, pokud chceme představit vlastní pohled na činnosti a požadavky v produktu. K tomu, abychom rozvíjeli myšlenky o produktu, používá se snadno srozumitelný jazyk, tak aby to bylo pro všechny jasně srozumitelné. A tak se můžou zapojit aktivně do procesu testování lidé, kteří mají na starosti obchodní analýzu a správu produktu.

Přestože nástroje BDT jsou speciálně vyvíjeny pro použití v projektech BDD (Behavior Driven Development), lze tyto nástroje brát jako speciální formu nástrojů, které pomáhají v TDD.

Dan North, který vyvinul BDD, zjistil, že jednotkové testy by měly být pojmenovány celými věty, které by měly začínat slovy „Měl by“ (z anglického slova should) a měly by být psány v pořadí dle byznys významu. Akceptační testy by měly být psány po vzoru uži-

vatelských příběhů agilního frameworku. Předloha: *„Jako [role uživatele] chci [nějaký výsledek] abych [nějaký důvod].“* Akceptační kritéria by měly být psané v podmínkách scénáře a měly by být implementovány jako třídy. Předloha: *„Vzhledem k [počáteční souvislost], pokud [nastane událost], tak [zajisti nějaké výsledky]“*. (24) (25)

4.2 Používané nástroje pro automatizované testování webu

Testovacích nástrojů existuje celá řada. Proto bych rád zmínil jen ty nejpoužívanější.

4.2.1 Selenium

Pro vývojáře a testery, kteří mají nějaké zkušenosti s programováním a skriptováním, Selenium poskytuje flexibilitu, která u mnoha ostatních nástrojů není. Testy lze psát pomocí širokého spektra programovacích jazyků. Selenium je podporováno v Javě, C#, Python, PHP, Ruby, Perl a Groovy. Spustit testy lze na Windows, Linux, Android i iOS. Prohlížeče, ve kterých je možnost testovat, je taktéž široký výběr. Jde o Chrome, IE, Firefox, Opera, Safari a jiné.

Selenium má však nevýhodu v tom, že použití tohoto nástroje vyžaduje pokročilé programovací znalosti a u začínajících projektů také určitý čas k vytvoření vhodného Frameworku pro svůj produkt. Existují však již vytvořené Frameworky, které lze použít. (26) (27)

4.2.2 Katalon Studio

Katalon umožňuje vytvářet automatizované testy skriptováním, manuálně vybíráním klíčových slov, nebo způsobem „nahraj a zopakuj“. Dále umí vytvářet testovací výstupy a reportovat jejich výsledky. Nástroj je zdarma ke stažení po registraci na jejich stránkách. Použití vypadá uživatelsky přívětivé a existují video tutoriály, podle kterých lze začít. (28)

4.2.3 Unified Functional Testing (UFT)

Jedná se o oblíbený komerční nástroj pro automatizované testování desktopových, webových a mobilních aplikací. Používat UFT může také neodborný tester. Poskytuje funkci „nahraj a proveď“. K automatizaci aplikací používá VBScript. Umí velmi dobře

identifikovat objekty. Roční cena produktu je 3200 dolarů, což je v přepočtu asi 73 tisíc Kč. (26) (29) (30)

4.2.4 TestComplete

TestComplete také podporuje automatizaci testů pro desktop, web a mobilní aplikace. Poskytuje funkci „nahraj a proveď“. Testy lze pouštět na různých prohlížečích. Cena licence začíná v přepočtu kolem 72 tisíc Kč. (26) (31)

4.2.5 SoapUI

Kromě testování webu a mobilních aplikací zvládá testovat API a služby. Nabízí se jak open-source, tak i pro verze. (26) (32)

4.2.6 Robot Framework

Robot Framework se stal velmi populární díky své jedinečné syntaxi, která je velmi srozumitelná i pro méně zdatné jedince v programování.

Příklad syntaxe:

```
*** Test Cases ***
Valid Login
    Open Browser To Login Page
    Input Username    demo
    Input Password    mode
    Submit Credentials
    Welcome Page Should Be Open
    [Teardown]    Close Browser
```

Dále umí vytvářet přehled výsledků. Robot Framework je open source. (33)

4.2.7 Ranorex

Podporuje automatizaci testů pro desktop, web a mobilní aplikace. Je také vhodný i pro začátečníky, díky možnosti nahrávání postupů. Cena licence začíná okolo 59 tisíc Kč. (34) (35)

II. PRAKTICKÁ ČÁST

5 VÝBĚR NÁSTROJŮ A METOD

5.1 Webová aplikace

Vzhledem ke svým vlastním zkušenostem jsem si vybral programovací jazyk C#, se kterým mám již delší zkušenosti. Pro jazyk C# existuje ASP.NET Framework, který je určený k vytváření webových aplikací. ASP.NET je nástupcem ASP. Konkrétně jsem si vybral ASP.NET 4.5 MVC Framework.

5.1.1 ASP.NET

Jedná se o vyspělou architekturu. Poskytuje služby nezbytné k sestavení na byznys úrovni. Malou nevýhodou může být, že produkt napsaný v tomto Frameworku lze nasadit pouze na servery s Windows. V případě, že by byl požadavek, aby výsledná aplikace mohla běžet na serveru s Linuxem , nebo macOS, bylo by potřeba zvolit ASP.NET Core. Nicméně .NET Framework nabízí vytvoření webových formulářů a webových stránek. ASP.NET obsahuje knihovny, které výrazně ulehčí vývoj samotné aplikace. Tyto knihovny obsahují hotová řešení pro bezpečnost, autentifikaci uživatele, práci s databází a jiné. (36)

5.1.1.1 Dynamický web

ASP.NET Framework poskytuje možnost vytvářet dynamické webové stránky. To funguje tak, že server ještě před tím, než odešle data uživateli, má možnost ještě stránku upravit. Oddělení logiky od prezentace je velmi populární. Usnadňuje to údržbu a zpřehledňuje aplikaci. Aktualizace informací na stránce je mnohem snadnější, než je tomu u statických webů.

Díky tomu lze vytvářet propracovanější grafické rozhraní. Lze například vytvářet rolovací menu, které reaguje na pohyb kurzoru, změnit, nebo rozšířit obsah stránky pro identifikované návštěvníky a jiné.

Zpracování takových stránek probíhá tak, že prohlížeč odešle požadavek na webový server o dynamickou stránku. Server stránku vyhledá a předá ji aplikačnímu serveru. Aplikační server vyhledá na stránce případné instrukce, které zpracuje, a stránku doplní, tak aby odpovídala požadavkům. Aplikační server dokončenou stránku předá webovému serveru, který ji pak následně předá prohlížeči, který si o ni požádal.

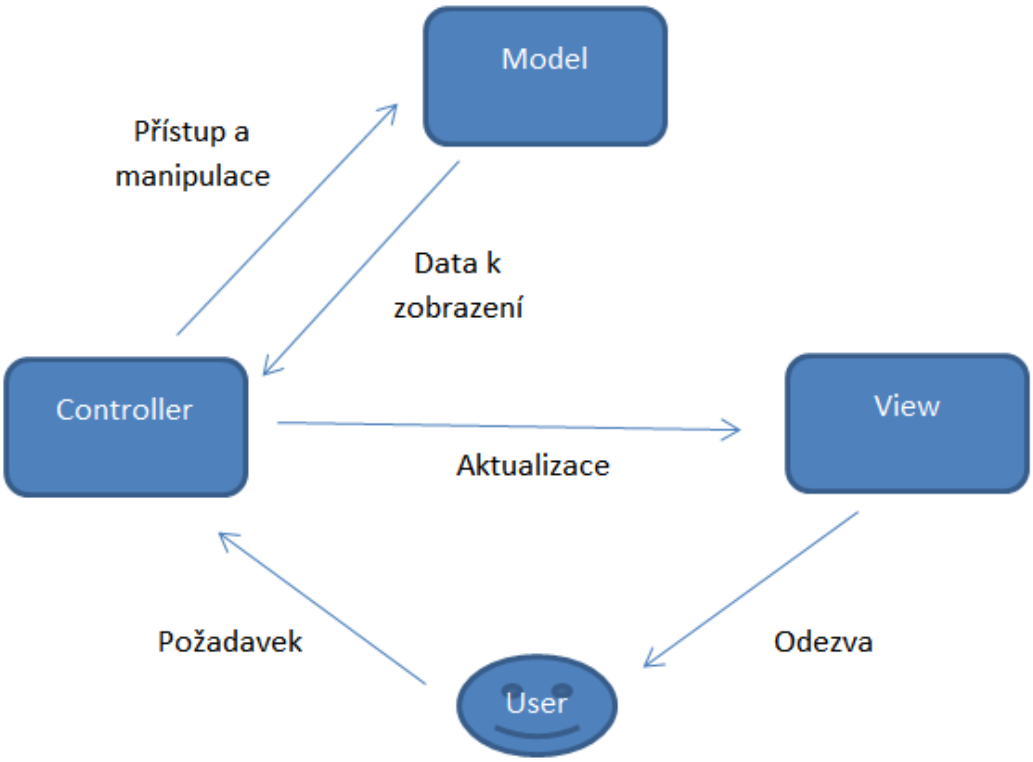
Dalším důležitým poznatkem je přístup k databázi. Aplikační server nemůže s databází komunikovat přímo. Je potřeba využít databázový ovladač. Pomocí tohoto ovladače má možnost aplikační server vytvořit dotaz na databázi. Databáze dotaz zpracuje a vrací sadu záznamů. V databázi lze shromažďovat data, která při zpracování dynamických stránek aplikační server později vyhledá v databázi a vrátí stránku s daty, která odpovídají požadavkům.

Vytváření rozhraní pro uživatele je tak daleko příjemnější. Stránky, které se ve výsledku zobrazí uživateli, lze vhodně rozdělit na několik částí. Tedy například navigační menu a hlavičky lze napsat do jednoho souboru, který popisuje rozvržení stránky, a obsah stránky, který se mění dle aktuální pozice na webu, lze napsat zvlášť do samostatných souborů, které obsahují jen jádro samotné stránky bez navigačního menu a hlaviček. Dále lze vytvářet jen části stránek, které lze pak použít na jednotlivých stránkách. Celý vývoj je pak daleko přehlednější a změny, které především v počátku probíhají (např. změna názvů), zaberou jen minimum úprav. (37)

5.1.2 ASP.NET MVC Framework

Model-View-Controller (MVC) je architektura, která rozděluje do tří samostatných, nezávislých komponent datový model aplikace, uživatelské rozhraní a řídicí logiku. Úprava jedné z komponent ovlivní zbylé jen minimálně.

Obecný princip MVC je vyobrazený na obrázku (Obrázek 5: MVC model). Uživatel provádí nějakou činnost v uživatelském rozhraní (View). Řadič o činnosti dostane informaci a přistupuje k modelu. Model na základě požadavku provede úkon s databází a vrátí do řadiče případná data. Řadič zpracuje získaná data a vytvoří uživatelské rozhraní. (38) (39)



Obrázek 5: MVC model

Model odpovídá za obchodní logiku, všechny funkce, které lze provádět s modelem a uchovává si stav aplikace. Pro složité uživatelské rozhraní se používá ViewModel (model pro zobrazení), který controller vytváří a plní obdrženými daty pro daný model.

Uživatelské rozhraní odpovídá za prezentovaný obsah. Aby bylo možné vkládat .NET kód do HTML, používá se zobrazovací modul Razor. Logika by zde měla být minimální a měla by se týkat pouze zobrazování obsahu.

Kontrolér odpovídá za zpracování činnosti uživatele, práce s modelem a za výběr zobrazení, které má být vykresleno. Kontrolér je v MVC vstupní bod, který vybírá typ modelu, se kterým se bude pracovat a které zobrazení se vykreslí. (40)

5.2 Testovací framework

Pro část testování jsem si vybral Selenium, konkrétně Selenium WebDriver. Pro mou potřebu nabízí všechny důležité funkce. Podpora pro tento nástroj je velmi obsáhlá a komunita používající tento nástroj vytvořila nespočet příspěvků, kde si lze některé dobré rady

nastudovat. Další nezanedbatelný důvod je ten, že Selenium jsem používal celou svou praxi při testování webových aplikací.

6 NÁVRH APLIKACE

Při návrhu aplikace jsem se snažil zohlednit známé aspekty, které s problematikou souvisí a promyslet, které komponenty budou nezbytně potřebné.

6.1 Identifikace uživatele

První část, bez které by se aplikace nedala vytvořit, je identifikace uživatele. Díky identifikaci lze dosáhnout jistého zabezpečení dat, tím, že data, která jsou jednoho uživatele, neuvidí ostatní.

6.1.1 Role

Aby mohla vzniknout administrační sekce v aplikaci, je vhodné nastavit uživatelům role. Role umožňují skupinově rozhodovat o tom, kdo má povolení provádět jednotlivé akce. Jako základ budou použity dvě role a to role pro admina a role pro uživatele, kteří jsou zaregistrovaní. Nepřihlášený uživatel bude bez skupiny a jeho práva budou tedy minimální.

6.1.2 Přihlášení a registrace

K identifikaci uživatele je potřeba, aby měl uživatel možnost registrovat se a přihlásit. Pro ověřování je zapotřebí uchovat informace o uživatelích včetně hesla. Heslo však musí být chráněno před hrozbou útoku, proto je potřeba jej vhodně zašifrovat. Vhodné je použít třeba funkci Hash. Mimo možnost registrovat se klasickým způsobem (uživatelské jméno a heslo), jsem využil velmi oblíbenou možnost pro přihlášení se pomocí účtu google. Jak tuto možnost implementovat Microsoft popisuje na svých Docs stránkách (https://docs.microsoft.com/cs-cz/aspnet/web-api/overview/security/external-authentication-services#GOOGLE).

6.2 Návrh struktury

Aplikace se má zabývat testováním. Hlavním zkoumaným prvkem bude tedy test. Test jakožto takový má nějakou strukturu. Má název, podle kterého by uživatel mohl odvodit, co daný test provádí. Dále by mohl mít popis, kde by bylo přehledně popsáno, o jaký test jde. Každý test má nějaký děj, takže bude obsahovat kroky, které budou určovat děj testu. Těmito kroky jsou myšleny jednotlivé akce, které lze na webové stránce provádět

(např. kliknutí na element). Každý test také musí začít otevřením prohlížeče na určené stránce, proto bude třeba uchovat odkaz.

Tyto testy je vhodné přiřadit vzhledem k nějakému projektu. Tímto projektem můžeme myslet třeba testovanou aplikaci. V rámci jednoho projektu se vytváří množství testů, tak aby bylo možné otestovat aplikaci co nejúčinněji. Mimo to v rámci jednoho projektu je běžné, že pracuje více testerů. Proto je potřeba aby byla možnost přidávat další členy týmu do projektu.

Je vhodné si uvědomit, že každý uživatel, může být součástí více projektů. Proto by měl mít možnost vytvořit si více projektů, a být přidán nejen do jednoho projektu.

6.2.1 Tvorba testů

Z kroků v testu by měl být čitelný postup, co se děje v testu. V ideálním případě by napsáním testu měl být samotný test case (testovací scénář). Čímž odpadá nutnost udržovat ještě dokumentaci testovacích scénářů. Na příklad by to mohlo vypadat následně:

Student UTB má přístup do STAG

1. *Otevřít stránku „https://stag.utb.cz/portal“*
2. *Napiš „jmeno_uzivatele“ do „Uživatelské jméno:“*
3. *Napiš „heslo_uzivatele“ do „Heslo:“*
4. *Klikni na „Přihlásit se“*
5. *Je zde viditelný element „Zobrazit můj rozvrh“*

Po rozboru toho, jak psát správně testy, jsem si vytvořil představu, jak uživateli přehledně a jednoduše poskytnout možnost psaní testovacích kroků. Každý krok musí vykonávat nějakou činnost. Tato činnost je prováděna nad nějakým elementem. Příkladem je třeba „Kliknutí na tlačítko ‘Potvrdit’“. Některé akce vyžadují ještě další atribut. Například akce pro zapsání textu do elementu. Zde je atributem hodnota, kterou je požadováno vepsat do elementu. Tyto kroky musí být prováděny ve správném pořadí, což je také důležité zohlednit.

6.2.2 Elementy

U elementů je třeba uchovávat informace o identifikátoru. Selenium umí několika způsoby hledat elementy na stránce. Osobně jsem si zvolil CSS selektory, se kterými mám

pozitivní zkušenosti. Občas je sice nemožné přesně určit element jen pomocí CSS selektoru, ale při správně navrženém webu by se tato situace neměla nastat.

Element by měl mít také nějaké pojmenování, které bude dávat smysl běžnému čtenáři. Jde o hodnotu, která se bude zobrazovat uživateli při výběru elementu, ale také v logování testů.

Element lze používat v rámci jednoho projektu a to v libovolném množství testů. Bylo by nevhodné vytvářet v každém testu selektor, který byl již jednou vytvořen. V případě změny názvu, nebo selektoru by tak bylo třeba upravovat všechny testy, ve kterých byl element použitý.

6.2.3 Testovací logy

Užitečnou součástí automatizovaného testování jsou logy, ze kterých lze vyčíst, jak test dopadl. V případě, že některý krok nemohl být proveden, bude toto zaznamenáno v takovém logu i s případnou zprávou, která popisuje tuto událost. Díky logům má tester možnost vidět, co přesně se stalo v testu. Jestliže tedy test selže (nalezne chybu v testované aplikaci), je pro testera mnohem snazší reportovat tuto skutečnost. Tester nemusí vytvářet manuálně postup, jak reprodukovat chybu, ale stačí, aby předal vytvořený log. Vývojář pak může projít tento scénář vlastnoručně a analyzovat tak chybu, která při vývoji vznikla. V ideálním případě tester nemusí předávat tuto zprávu manuálně, ale vývojář má možnost sám přistoupit k těmto výsledkům, což zefektivní celý proces.

U logování je stejně důležité, jako u tvorby testů, aby byly logy čitelné a pochopitelné i pro lidi, kteří zastupují stranu zákazníka. Mnohdy jsou to lidé, kteří jsou bez technických znalostí.

Logy by také měli odpovídat 1:1 krokům v testovacím scénáři. Proto jsem navrhl, aby takový log měl stejnou strukturu jako vytváření testů, jen doplněné informací, zda byl krok úspěšný.

Dále je důležité zahrnout informace související se spouštěním testů. Konkrétně tedy informaci o čase, kdy test byl spuštěn. Tato informace je velmi důležitá především u projektů, které jsou velmi rychle vyvíjeny a i konkrétní minuta rozhoduje o tom, v které fázi jsme aplikaci testovali. Díky tomu je snazší určit, která změna měla za vinu rozbití funkcionality. Také je třeba udržovat informaci o tom, na kterém prostředí test proběhl. Chyby

mohou být totiž závislé na prostředí. To co funguje například v Chrome prohlížeči na systému Windows 7, nemusí nutně fungovat v prohlížeči Edge na systému Windows 10.

6.2.4 Ukládání dat

Pro ukládaní jsem jednoznačně zvolil databázi. Vzhledem k velkému množství dat a také k možnostem, které databáze nabízí. Například relace mezi daty, díky kterým lze vhodně a rychle dohledat potřebné informace, které jsou závislé na záznamech z jiných tabulek.

.NET technologie poskytuje možnost využít Entity Framework, který jsem taktéž použil. Tento Framework umožňuje pracovat s databází s použitím objektů napsaných v .NET. Entity Framework mapuje relace objektů. Díky Entity Framework není třeba vytvářet většinu kódu pro přístup k datům, který by jinak vývojář musel psát. (41)

6.2.5 Spouštění testů

Testy potřebují být spouštěny v nějakém připraveném prostředí. Často se pro tyto účely vytváří speciálně virtuální počítač ke vzdálenému připojení. Díky virtualizaci máme možnost nasimulovat prostředí, které odpovídá reálnému produkčnímu prostředí. Na toto prostředí se tedy musí nainstalovat potřebné aplikace, aby bylo možné testy vzdáleně pouštět.

K tomuto problému jsem přistoupil tak, že jsem navrhl, aby byla vytvořena aplikace, která bude mít na starost spouštění testů. Tato aplikace bude aktivní na prostředí s nainstalovanými prohlížeči a bude stále aktivní, aby bylo možné kdykoliv vyvolat akci, která testy spustí. Tato aplikace dostane informaci o tom, který test spustit. Následně si aplikace načte data, o tom, co se má a jak testovat. Poté spustí test a během testování bude odesílat průběžné výsledky k uložení do databáze.

Pro komunikaci mezi touto aplikací a severem, na kterém běží webová aplikace, jsem navrhl implementaci modelu klient-server. V tomto případě bude serverem aplikace, která má na starosti spouštění testů. Na tento server bude webová aplikace odesílat požadavky ke spuštění testu tak, že webová aplikace spustí klienta, který naváže komunikaci se serverem (aplikace ke spouštění testů) a odešle informaci. Po té se klient může ukončit. Neboť další komunikace již probíhat nemusí.

Pro ukládání informací o průběhu testu lze využít možnost odesálání požadavků na kontrolér přes API. Tento kontrolér již zpracuje požadavek a provede činnost vedoucí k uložení dat.

Jelikož průběh testu je navržen tak, aby byly ukládány jako posloupnost kroků, bylo třeba navrhnout také speciálně engine programu, aby uměl tato data zpracovat a řídit běh testů přesně podle požadavků. Zde bylo třeba navrhnout architekturu tak, aby bylo možné předávat parametry absolutně univerzálně.

6.3 Design

K navržení designu jsem využil nástroj Draw.io, který je online a je zdarma k použití. Navrhl jsem zde několik stránek, které jsem považoval za nezbytné. Tyto návrhy mi pomohly promyslet všechny důležité kroky, se kterými jsem se posléze setkal při budování výsledného designu. Mým cílem bylo vytvořit design jednoduchého stylu, který bude přehledný a uživatelsky přívětivý. Webová aplikace by měla být intuitivní, aby nevyžadovala speciální zaškolení, nebo tvorbu dokumentu, dle kterého by se uživatel musel řídit. Snažil jsem se navrhnout stránky tak, aby bylo i začátečníkům, nezkušeným uživatelům, jasné, co je od uživatele vyžadováno a aby se mohl snadno zorientovat. Také aby byly reporty a výsledky testů pro něj snadno čitelné a pochopitelné.

Návrh designu, který jsem vytvořil v Draw.io, jsem přidal do přílohy (PŘÍLOHA P I: NÁVRH DESIGNU).

7 IMPLEMENTACE APLIKACE

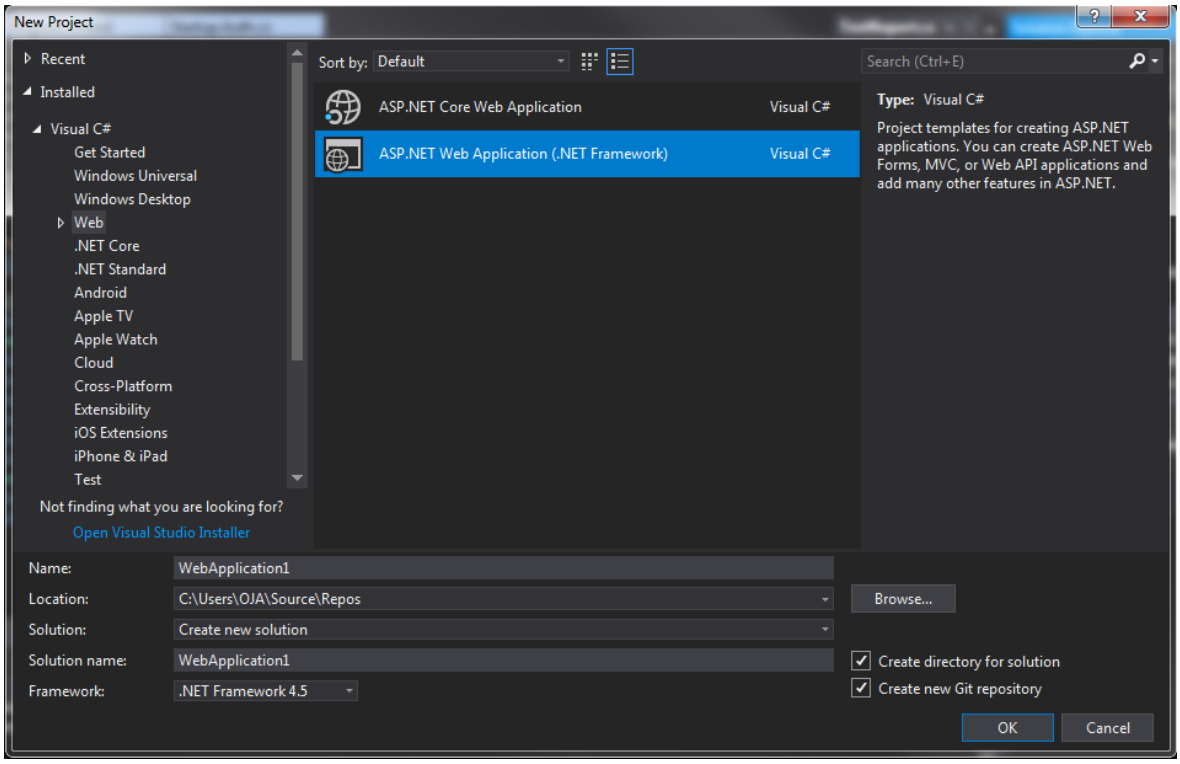
7.1 Prostředí pro vývoj

Pro tvorbu aplikací na platformě .NET je potřeba prostředí Microsoft Visual Studio. Osobně jsem využil verzi Microsoft Visual Studio Community 2017, která je zdarma při dodržení licenčních podmínek uvedených na stránkách Microsoft (https://visualstudio.microsoft.com/cs/license-terms/mlt553321/). Tato licence osahuje zcela všechny funkce a nástroje, které pro vývoj mé aplikace budu potřebovat.

Ještě než jsem mohl začít s vytvořením projektu, bylo třeba doinstalovat některé komponenty do Visual Studia. K tomu slouží Visual Studio Installer. Jedná se o vývojové nástroje pro ASP.NET a web, .NET Framework, ve kterém chcete pracovat (já si zvolil 4.5) a ASP.NET MVC.

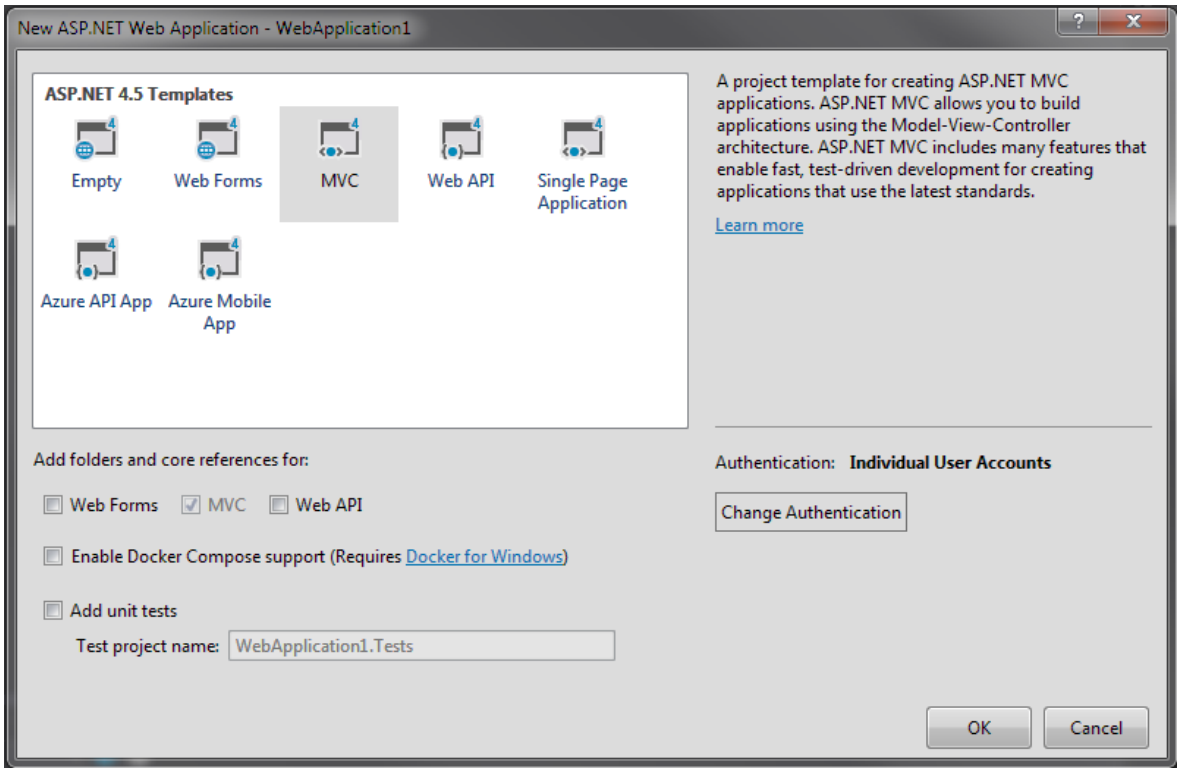
7.2 Vytvoření projektu

Pro vytvoření nového projektu v aplikaci jsem vybral šablonu ASP.NET Web Application (.NET Framework) (viz Obrázek 6: Výběr šablony pro vytvoření projektu).



Obrázek 6: Výběr šablony pro vytvoření projektu

Po potvrzení výběru šablony se objeví ještě jedno okno pro výběr šablony ASP.NET 4.5. Zde jsem zvolil MVC a autentizaci jsem změnil na „Individual User Accounts“ (viz. Obrázek 7:Výběr šablony MVC), aby bylo možné si vytvořit přihlašování uživatelů.



Obrázek 7:Výběr šablony MVC

Visual Studio vytvoří projekt, který je již možné spustit. Již neupravená šablona vypadá celkem přijatelně i z designového pohledu. Z tohoto důvodu jsem vycházel i ve vytváření vzhledu webu z této šablony, kterou jsem upravoval do vlastních představ.

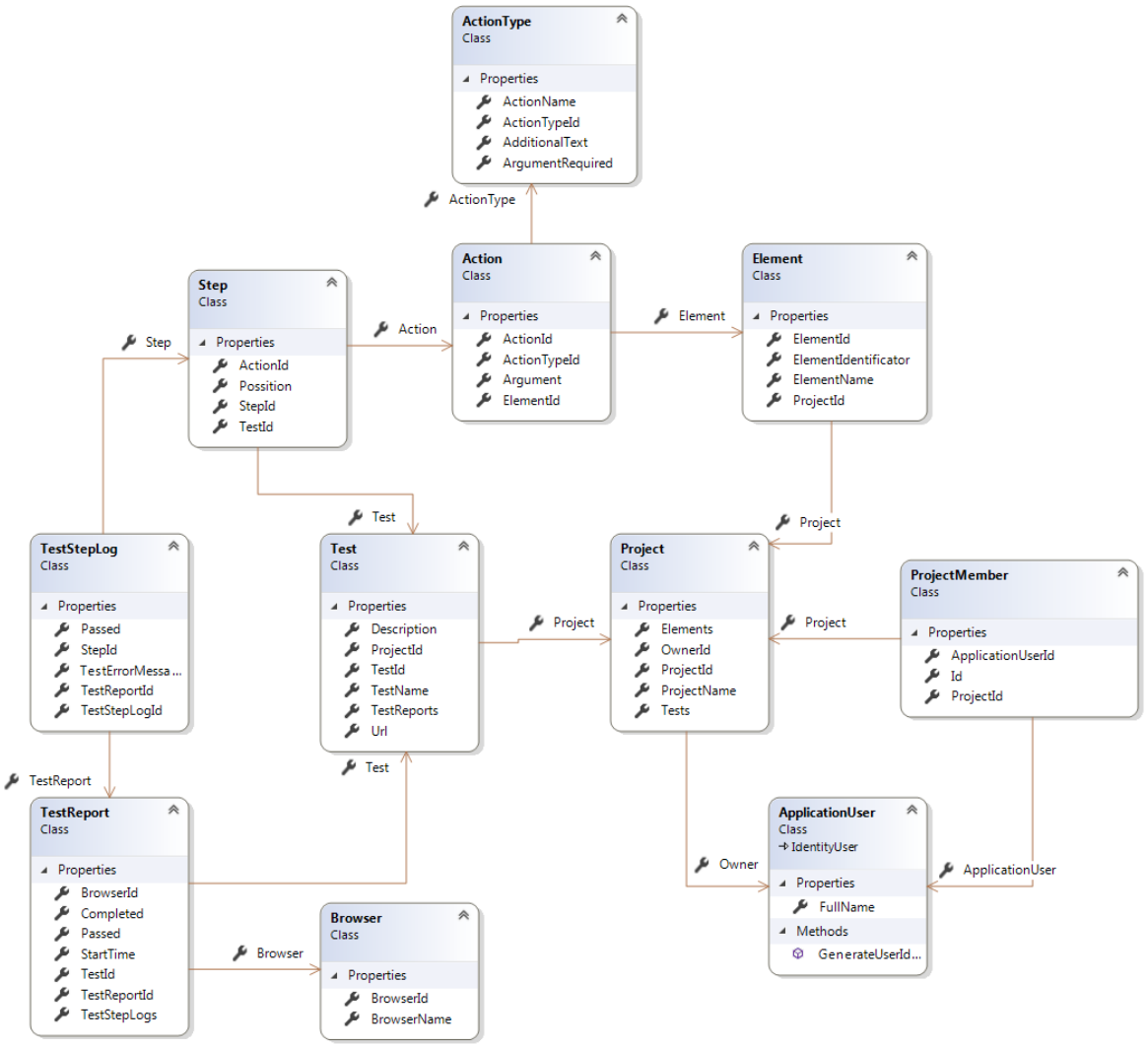
MVC jsem si zvolil, jelikož pro mou implementaci poskytuje výhody popsané v teorii. Při implementaci požadovaných funkcí, mi poskytuje MVC možnost oddělit návrh modelu, business logiku a vzhled stránky.

7.3 Instalace balíčků

Aby bylo možné využít knihovny, které pomáhají při vývoji aplikací, je třeba si nainstalovat je v rámci balíčků NuGet. První balíček, který jsem potřeboval byl balíček EntityFramework od Microsoft. Tento balíček umožňuje využít možnosti Entity Frameworku pro práci s databází.

7.4 Modely

Na základě návrhu jsem si vytvořil objekty, které jsou potřeba pro správné fungování aplikace. Tyto modely jsem zobrazil v Obrázek 8: Diagram tříd použitých modelů.



Obrázek 8: Diagram tříd použitých modelů

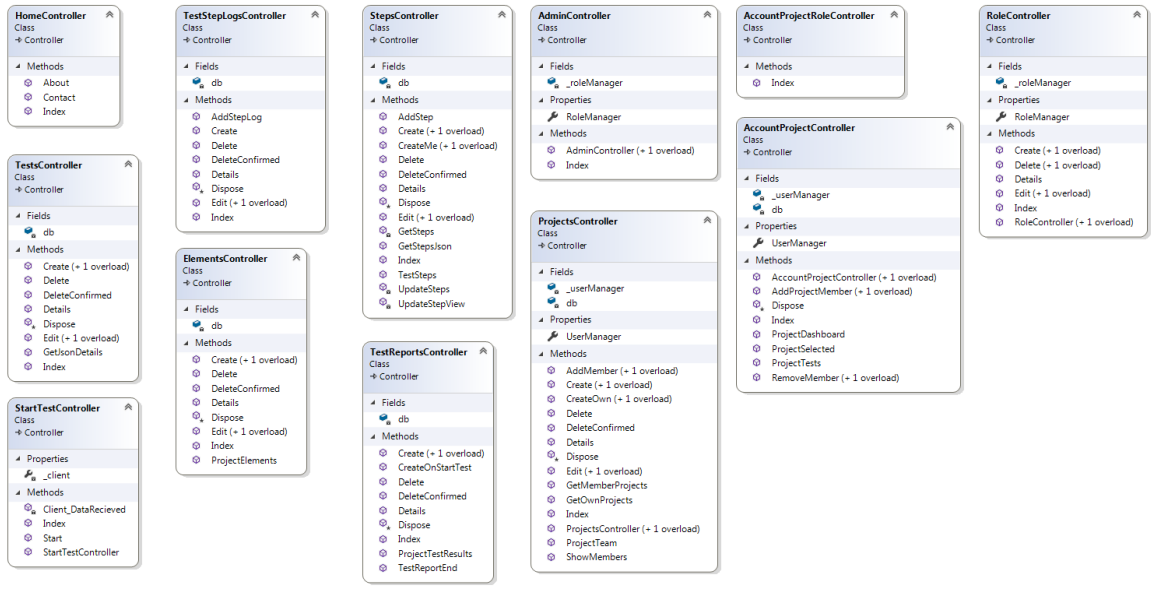
U modelů je důležité myslet na to, aby obsahovaly všechny důležité parametry včetně ID, jelikož by nám Entity Framework nemohl vytvořit relaci s databází. Dále je potřeba, aby modely, které mají vztah s jiným modelem (např. model *Step* má vlastnost *Test*, což je další model) měli i vlastnost ID pro vytvoření vazby k modelu, se kterým takový vztah tvoří. Jako příklad použiju Obrázek 9: Třída Step, kde jsem okomentoval řádky, vedoucí k pochopení problematiky.

```
namespace Diplomka.Models
{
    [Table("Steps")]    //Atribut pro vztvoření databázové tabulky s názvem 'Steps'
    public class Step
    {
        // Atribut pro vytvoření primárního klíče
        [Key]
        //Atribut definující, že ID bude generováno automaticky
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int StepId { get; set; }

        //Atribut pro vytvoření cizího klíče, který je povinný a vztahuje se k vlastnosti 'Test'
        [ForeignKey("Test"), Required]
        public int TestId { get; set; }

        //Vlastnost test je pouze virtuální, jelikož v databázi bude použit cizí klíč k identifikaci testu
        public virtual Test Test { get; set; }
        [ForeignKey("Action")]
        public int ActionId { get; set; }
        public virtual Action Action { get; set; }
        [Required]
        public int Possition { get; set; }
    }
}
```

Obrázek 9: Třída Step

7.5 Kontroléry

Následně pro implementaci byznys logiky jsem si vytvořil kontroléry. Vytvořené kontroléry si lze prohlédnout v Obrázek 10: Diagram tříd kontrolérů.



Obrázek 10: Diagram tříd kontrolérů

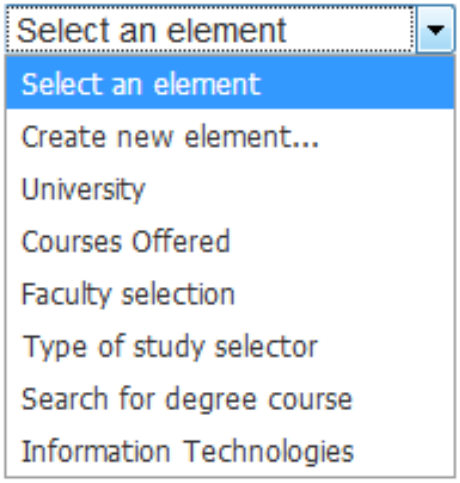
Zde je nejen třeba zohlednit byznys logiku, ale je třeba myslet i na to, co umožníme uživateli ve výsledku, aby viděl na stránkách (uživatelské rozhraní). Někdy je potřeba vytvořit prezentační modely. Tyto modely pak umožňují v uživatelském rozhraní (View) pra-

covat s daty, které jsme si v kontroléru připravili. Jedním takovým příkladem je v mé práci třeba *ElementSelectionViewModel* (viz Obrázek 11: Třída ElementSelectionViewModel).

```
namespace Diplomka.Models
{
    public class ElementSelectionViewModel
    {
        public ElementSelectionViewModel() { }
        public ElementSelectionViewModel(List<Element> elements)
        {
            _elements = elements;
        }
        private readonly List<Element> _elements;

        [Display(Name = "Selected Element")]
        public int? SelectedElementId { get; set; }
        public IEnumerable<SelectListItem> ElementItems
        {
            get
            {
                var items = new SelectList(_elements, "ElementId", "ElementName");
                return DefaultElementItems.Concat(items);
            }
        }
        private IEnumerable<SelectListItem> DefaultElementItems
        {
            get
            {
                return new List<SelectListItem>() {
                    new SelectListItem() { Text = "Select an element" },
                    new SelectListItem() { Text = "Create new element...", Value = "new element" } };
            }
        }

    }
}
```

Obrázek 11: Třída ElementSelectionViewModel

V tomto modelu jsem si vytvořil vlastnost *ElementItems*, která vrací prvky pro vytvoření listu s elementy. Zde jsem ještě přidal privátní vlastnost *DefaultElementItems*, díky které je možné do listu přidat prvky do listu, o které lze tento list rozšířit. Výsledný prvek, který lze díky této vlastnosti vytvořit je na Obrázek 12: Výběr elementu.



Obrázek 12: Výběr elementu

V kontroléru, je třeba pro danou funkci tento model předat jako návratovou hodnotu zabalenou do objektu *View*, případně *PartialView*. V mém případě jsem použil *PartialView*, neboť jsem se rozhodl, že výsledné zobrazení pro tuto funkci bude pouze jako součást jiného zobrazení. Tomu odpovídá i návratový typ *PartialViewResult*.

```
public PartialViewResult ProjectElements(int id)
{
    var elements = db.Elements.Where(e => e.ProjectId == id).ToList();
    var list = new ElementSelectionViewModel(elements);
    ViewBag.ProjectId = id;
    return PartialView(list);
}
```

Obrázek 13: Funkce pro získání elementů pro projekt, který je identifikovaný vstupní hodnotou id

7.6 View

Soubory pro grafická rozhraní v ASP.NET MVC Framework nesou příponu „.cshtml“. Tyto soubory jsou známé pod názvem Razor stránky. Syntaxe Razor nám umožňuje použít při vytváření html kódu pro výsledný vzhled stránky také funkce C#, díky čemuž máme možnost vkládat a předpřipravit výsledné HTML stránky, které se odešlou uživateli k zobrazení.

Pokud chceme vložit funkci, nebo proměnnou, musíme použít klíčový znak „@“. Za tímto znakem pak následuje požadovaný výraz. Například pro výpočet 1+1 bychom použili

výraz „@(1+1)“ a výsledkem by bylo, že by se před odesláním výsledného HTML souboru dosadila hodnota „2“ namísto použitého výrazu.

Pokud chceme na Razor stránce použít výstup z kontroleru, je potřeba zahrnout definici modelu na této stránce. To lze provést za pomocí klíčového slova „@model“, za kterým bude následovat reference na model. Pro lepší pochopení uvedu příklad z mé práce.

V kontroléru *ElementsController.cs* mám metodu *Create*, která vrací model *Element*. Na Razor stránce proto použiju „@model Diplomka.Models.Element“.

Grafické rozhraní se nachází ve složce Views, ve které se nachází další složky. Tyto další složky nesou vždy název odpovídající kontroleru. Jmenuje-li se kontroler, jako v mém případě *StepsController.cs*, název složky bude nést název odpovídající tomuto kontroleru jen s rozdílem, že se vynechává část „Controller.cs“, tedy má složka grafických rozhraní pro zvolený kontroler bude mít název Steps. Cesta ke grafickému rozhraní je dána vzorem *View/[název kontroléru]/[název metody]*. Cesta k mému grafickému rozhraní pro metodu TestSteps, která se nachází v kontroleru s názvem StepsController.cs, je tedy View/Steps/ TestSteps.cshtml.

Pokud však Razor stránka pro danou metodu ještě nebyla vytvořena, lze vytvořit pohled tak, že na metodu klikneme pravým tlačítkem myši a z nabídky vybereme možnost „Add View…“. Následně vyskočí okno, kde máme volbu pro vybrání šablony. Díky této možnosti se nám vygeneruje soubor, který je připraven ihned k použití bez dalších úprav. Pokud chceme však design dále přizpůsobit, není problém tak učinit. Dále v tomto okně, které nám vyskočilo, lze zvolit možnosti, jako je vytvoření souboru, který bude použitelný jen jako částečné zobrazení. V jiném případě lze použít možnost, že stránka bude používat layout stránku.

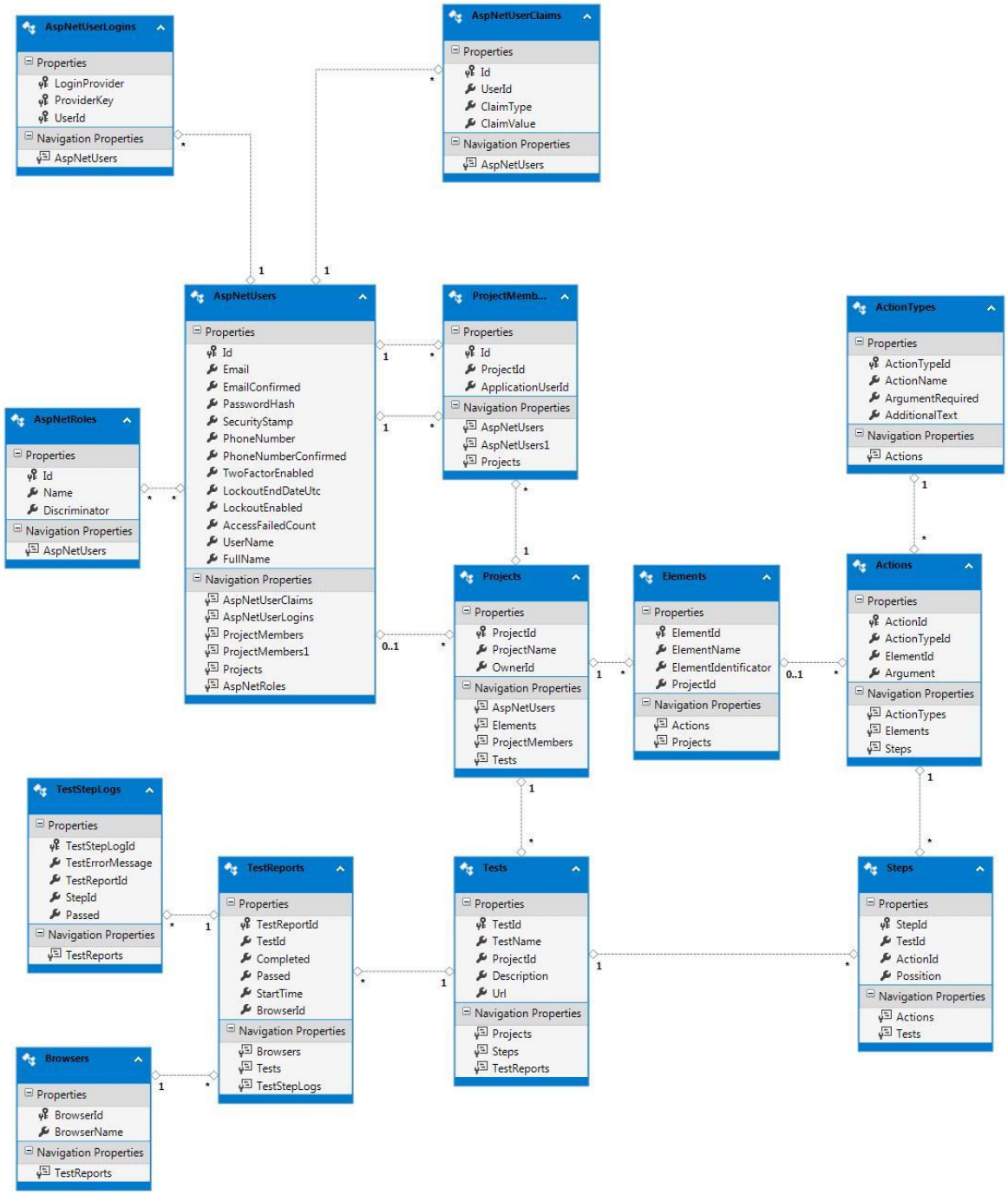
7.7 Update databáze

Před samotným spuštěním napsané aplikace je však potřeba vytvořit migrace pro nachystání databáze k použití a následně tuto databázi updatovat, aby se tato migrace provedla. Pro tento krok budeme potřebovat *Konzoli správce balíčků*. Tu lze najít ve VS v menu *Nástroje > Správce balíčků NuGet > Konzola správce balíčků*.

V této konzoli je nejprve potřeba povolit migrování. To lze provést zadáním příkazu *enable-migration*. Následně je potřeba vytvořit migraci a pojmenovat ji. Pro vytvoření *Initial* migrace použijeme příkaz *add-migration Initial*. Tento příkaz automaticky vygene-

ruje soubor, který obsahuje instrukce popisující činnosti, které se mají provést při update databáze. Souhlasíme-li s vygenerovanými instrukcemi, můžeme jít na další krok a tím je update databáze. Pro tuto akci použijeme příkaz update-database. Že byla databáze provedena úspěšně, poznáme z konzole, kde by se měla zobrazit zpráva „Running Seed method.“.

Pro mou aplikaci se vytvořila databáze, pro kterou jsem vytvořil diagram a ten zde vložil jako Obrázek 14: Diagram databáze.



Obrázek 14: Diagram databáze

7.8 Spuštění aplikace

Aplikace, kterou jsem začal vytvářet v rámci diplomové práce je stále ve fázi vývoje a tak je potřeba tuto aplikaci otevřít ve Visual Studiu (viz Prostředí pro vývoj42). Odsud bude možno aplikaci rovnou spustit v režimu *Debug*. Nejprve je potřeba spustit Projekt *Diplomka* a poté projekt *TestServer*.

Aplikace *Test Server* se otevře po spuštění projektu *TestServer*. Zde není třeba upravovat žádné údaje, neboť je aplikace stále ve vývoji. Je ovšem potřeba v aplikaci stlačit tlačítko *Start*, aby se spustila instance pro odposlech sítě. Ukázka aplikace je na Obrázek 15.Obrázek 1

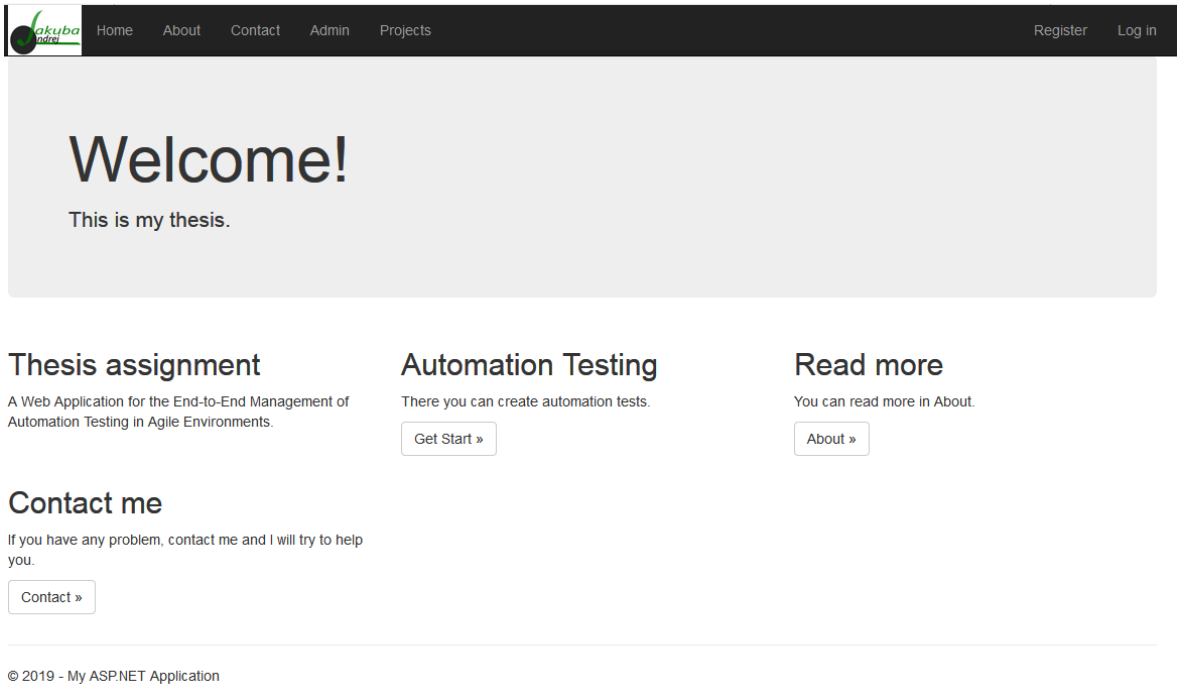
Obrázek 15: Spuštění aplikace Test Server

8 UKÁZKA APLIKACE

V této části provedu názornou ukázku, jak aplikace funguje.

8.1 Úvodní stránka

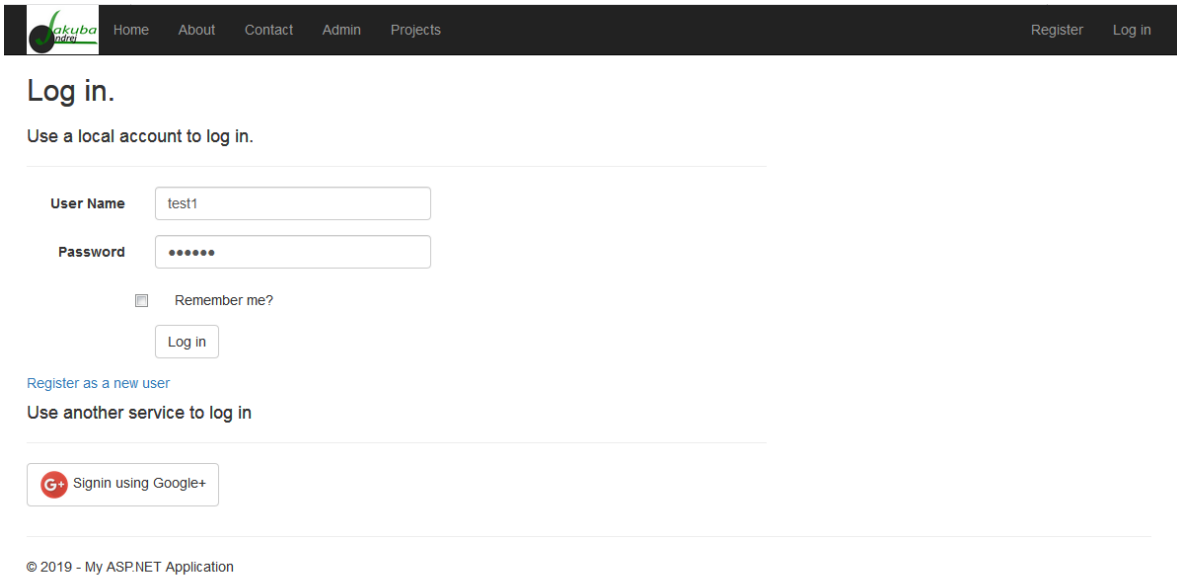
Po otevření aplikace se uživatel ocitne na úvodní stránce, kde nalezne základní odkazy.



Obrázek 16: Úvodní stránka

8.2 Přihlášení uživatele

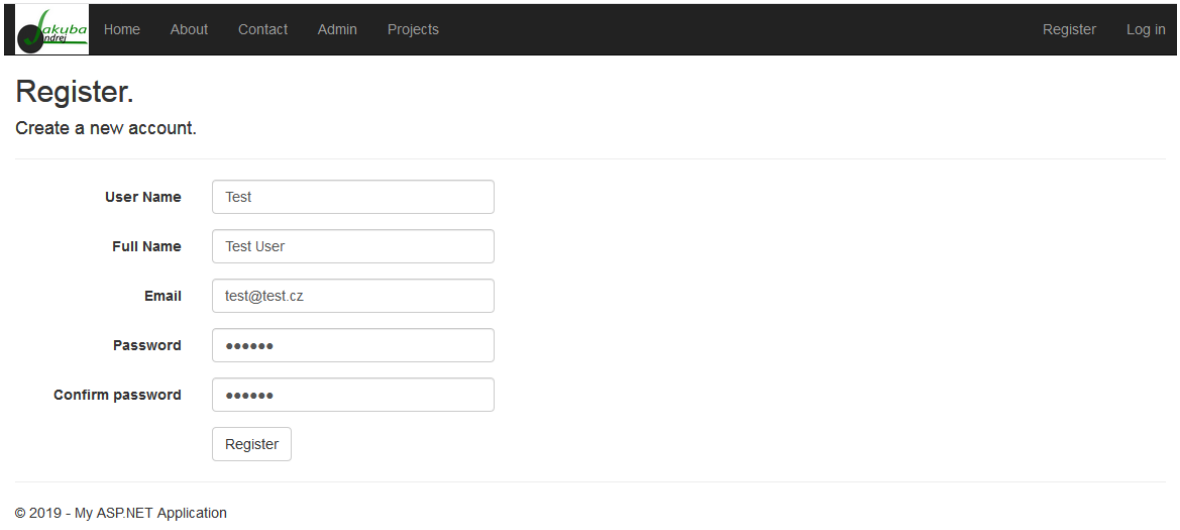
Pro přihlášení uživatele jsem vytvořil možnost jak přihlášení se klasicky, tak i pomocí účtu Google.



Obrázek 17: Přihlašovací stránka

8.3 Registrace uživatele

K tomu aby uživatel mohl pracovat v aplikaci je potřeba aby byl registrovaný, což se provede automaticky pokud se přihlásí pomocí Google účtu, nebo je potřeba aby vyplnil následující stránku.



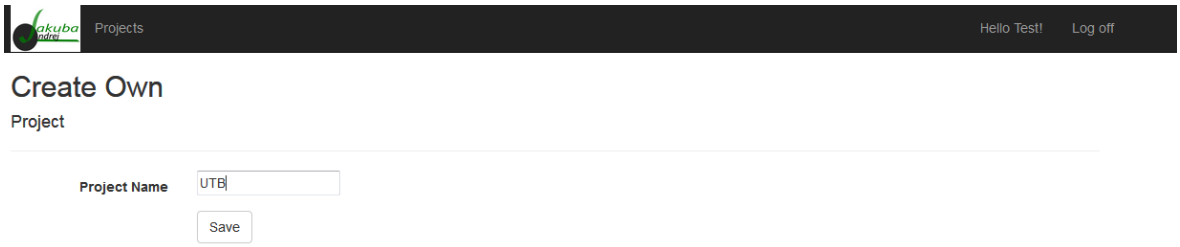
Obrázek 18: Stránka pro vytvoření uživatele

8.4 Vytvoření nového projektu

Po kliknutí na *Projects* v horním menu se načte stránka pro výběr projektu. Zde jsou všechny projekty, do kterých má přihlášený uživatel přístup. Ať už je uživatel vlastníkem projektu nebo jen člen týmu. Vzhledem k tomu, že jsem se nyní přihlásil jako nově registrovaný uživatel, tak zde nejsou žádné projekty. Pro vytvoření nového projektu zde slouží textový odkaz *Create new* (viz Obrázek 19).

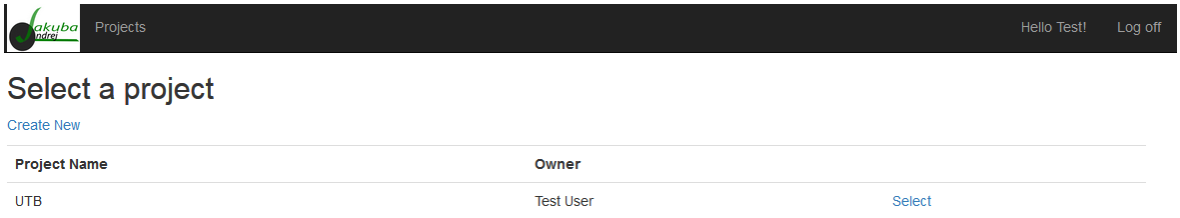
Obrázek 19: Výběr projektu

Po kliknutí na tento odkaz se načte stránka (viz Obrázek 20), kde je vyžadován název projektu. Po kliknutí na tlačítko *Save* se projekt vytvoří. Vlastníkem se stává automaticky uživatel, který tento projekt vytvoří.



Obrázek 20: Vytvoření nového projektu

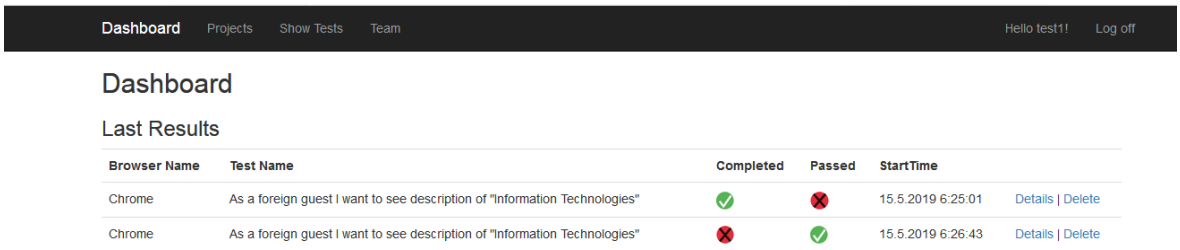
Po vytvoření projektu je uživatel automaticky přesměrován zpět na výběr projektu. Zde je již nový projekt vidět (viz Obrázek 21).



Obrázek 21: Projekt vytvořen

8.5 Otevření projektu

Projekt lze otevřít kliknutím na textový odkaz *Select* pro vybraný projekt na stránce pro výběr projektu (viz Obrázek 21). Po zvolení se uživateli zobrazí stránka s přehledem reportů, tedy s výsledky testů, které byly již spuštěny (viz Obrázek 22). Po zvolení projektu se taktéž změní výběr v horním menu.

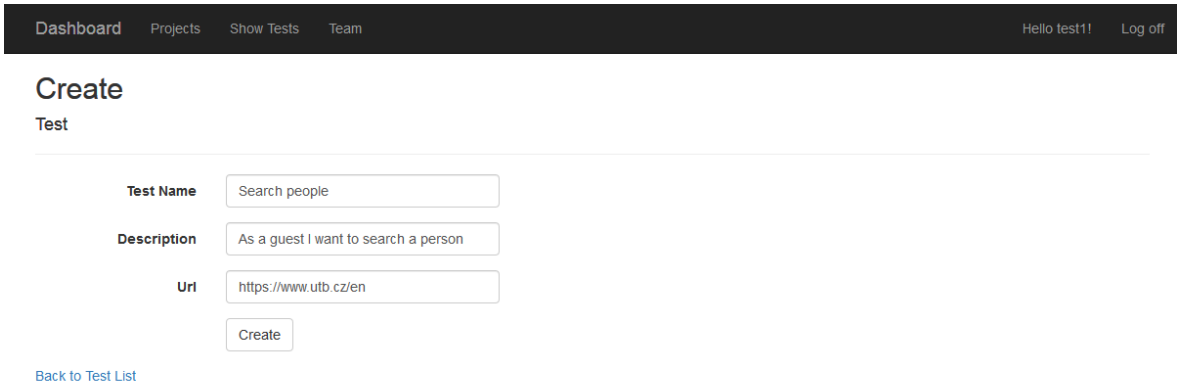


Obrázek 22: Dashboard

Ve výsledcích testů uživatel může ihned vidět, na kterém browseru byl test spuštěn, název testu, zda ten test již doběhl, zda ten test byl dokončen úspěšně a také kdy byl spuštěn.

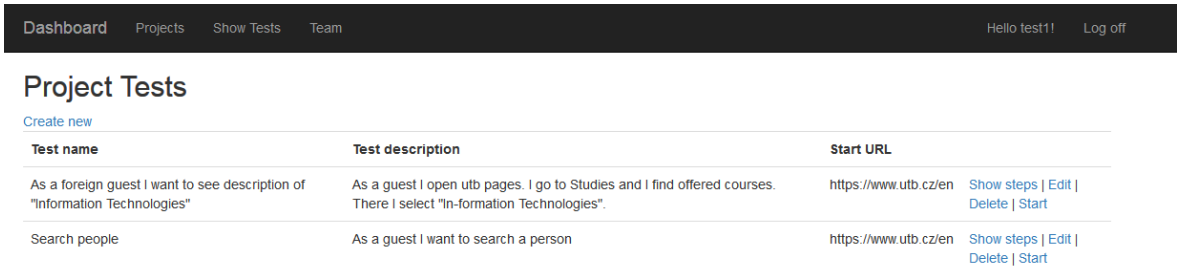
8.6 Vytvoření testu

Kliknutím na možnost *Show Tests* se uživatel dostane na přehled testů, které již ve zvoleném projektu existují. Pokud uživatel chce vytvořit nový projekt, lze tak učinit klepnutím na *Create new*. Uživateli se otevře stránka, kde je vyžadován název testu (je vhodné volit název, který vystihuje celý proces testu, i když by měl název obsahovat větu), popis testu (zde je dobré popsat celý postup, co se má testovat a další detaily) a odkaz stránky, na které test začíná (viz Obrázek 23). Po té stačí potvrdit tlačítkem *Create*. Po vytvoření je uživatel přesměrován zpět na přehled testů.



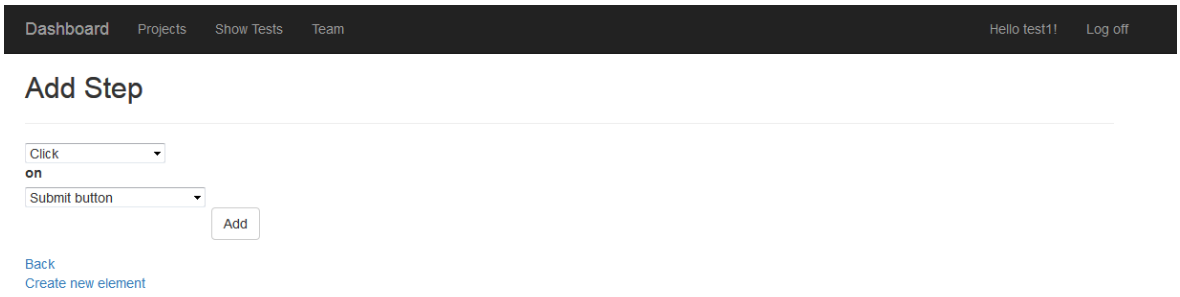
Obrázek 23: Přidání testu do projektu

To že byl test úspěšně přidán, pozná uživatel tak, že jej uvidí v přehledu testů (viz Obrázek 24). Nyní je třeba, aby uživatel do testu přidal kroky, které má test provádět. Bez těchto kroků test pouze otevře stránku a tím test končí. Kliknutím na *Show steps* u vytvořeného testu se uživatel dostane na stránku, kde má možnost kroky přidávat i odebírat.



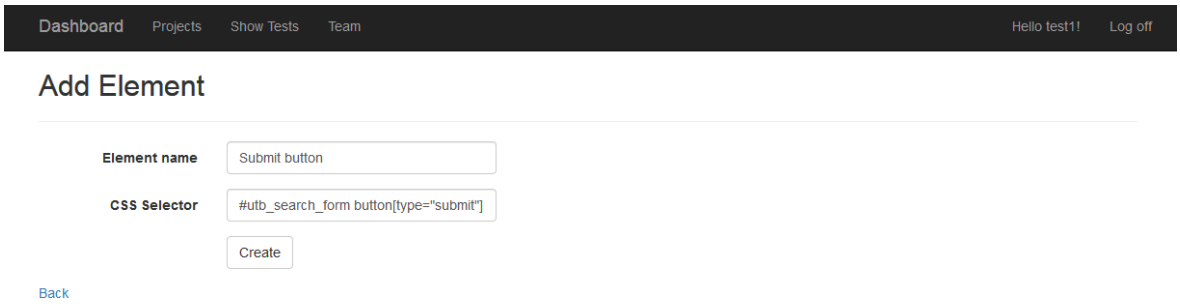
Obrázek 24: Přehled testů v projektu

Pro příklad jsem vytvořil test stránek UTB, kde chci najít svého vedoucího projektu dle příjmení. Pro přidání následujícího kroku je potřeba kliknout na *Add next step*. Zde má uživatel možnost vybrat druh akce, která má být vykonána v tomto kroku. Po výběru druhu akce se zobrazí zbytek parametrů, které je třeba vybrat pro vybranou akci. Jako příklad jsem použil akci kliknutí na potvrzovací tlačítko (viz Obrázek 25).



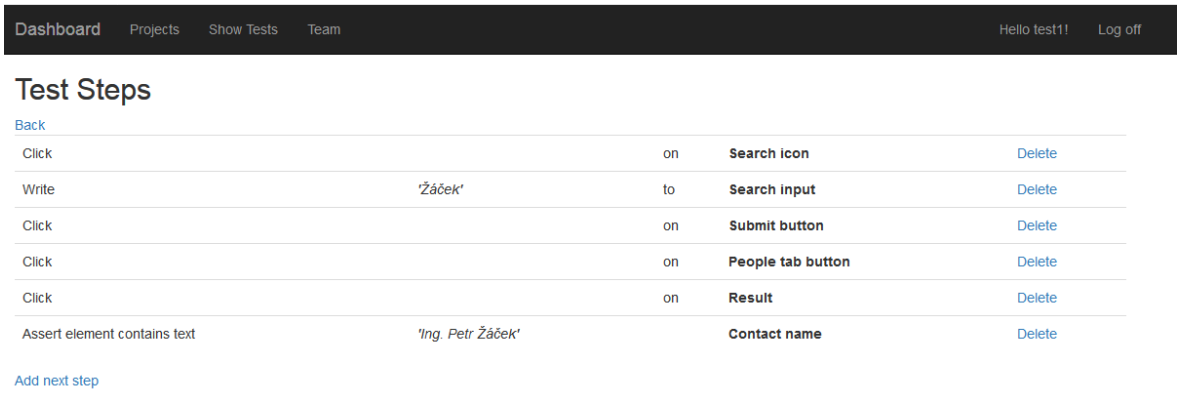
Obrázek 25: Přidání kroku do testu

K tomu abych mohl element pro potvrzovací tlačítko použít, musel jsem jej definovat. To jsem udělal tak, že na stránce pro přidání kroku jsem stiskl *Create new element*. Pro přidání elementu jsem tedy vložil vhodný název a CSS Selektor tlačítka (viz Obrázek 26).



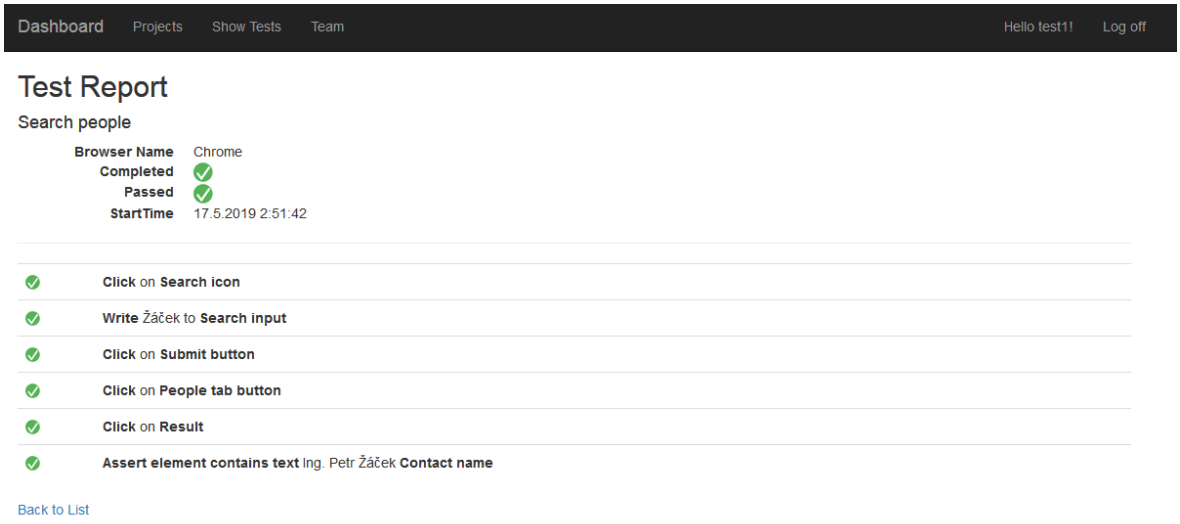
Obrázek 26: Přidání elementu

Tímto způsobem jsem postupoval, až jsem kroky popsal celý scénář, který chci otestovat (viz Obrázek 27).



Obrázek 27: Kroky testu

Po té co jsem test vytvořil a popsal pomocí testovacího scénáře pomocí kroků, jsem test spustil, abych ověřil jeho správnost. Po skončení testu jsem přešel na *Dashboard* Projektu a otevřel jsem si detail výsledku (viz Obrázek 28).



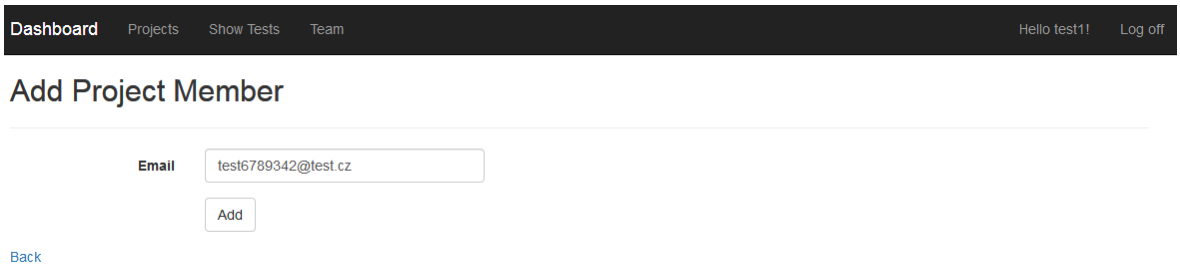
Obrázek 28: Výsledek spuštěného testu

8.7 Spuštění testu

Vytvořené testy lze spustit ze stránky s přehledem testů (viz Obrázek 24) kliknutím na *Start*. Testů lze spouštět i více v jednom okamžiku a to tolikrát, kolikrát uživatel stiskne právě *Start*. Osobně jsem spustil 4 testy v jednu chvíli a proběhly úspěšně.

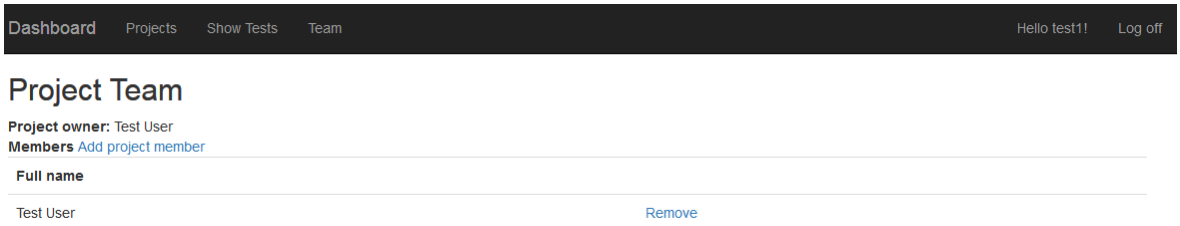
8.8 Přidání dalšího uživatele do projektu

Pro přidání dalšího uživatele do projektu je potřeba kliknout na *Team* v horním menu (uživatel musí mít vybraný projekt). Zde je možnost *Add project member*, na kterou je potřeba kliknout. Aby bylo možné přidat dalšího uživatele, je potřeba, aby tento uživatel byl zaregistrovaný v této aplikaci. Je-li uživatel registrovaný, lze jeho přidání provést zadáním jeho emailu a stiskem tlačítka *Add* (viz Obrázek 29).



Obrázek 29: Přidání uživatele do projektu

O tom, že je uživatel již součástí týmu se lze ujistit tak, že jej vidíme mezi členy týmu (viz Obrázek 30).



Obrázek 30: Tým na projektu

ZÁVĚR

Motivací pro tuto diplomovou práci mi bylo získat zkušenosti s tvorbou webové aplikace v technologii ASP.NET a možnost vytvořit aplikaci, na kterou jsem si dříve nenašel čas. Rád bych, aby aplikace pomohla při práci testerům, kteří nemají mnoho zkušeností s kódováním a umožnila jim tak testovat aplikace automatizovaně.

Při vývoji software je nedílnou součástí i testování kvality vyvíjeného software. Díky testování si můžeme být více jistí výsledkem, který se snažíme prezentovat zákazníkům. Odhalení chyby zákazníkem sice může být přínosné pro obě strany, nicméně pokud zákazník objeví chyb až příliš mnoho, byl by značně znepokojen.

V agilním prostředí tedy pro testování software používejme automatizovaných testů k pokrytí funkcionalit v dostatečné míře, aby nám tak testy šetřili čas i práci. Nesmíme však zapomínat i na testy manuální, které jsou vhodné například při testování nových funkcionalit.

Nástroje, které lze pro automatizaci testů použít, existuje mnoho. Důležité je shrnout si informace o tom, jak a co je potřeba testovat. Vhodnou volbou nástroje totiž můžeme výrazně ovlivnit vlastní efektivitu práce.

Implementace mé aplikace se částečně liší od návrhu. Tato skutečnost je dána mými poznatky, které přicházely až s tvorbou samotné aplikace. V zásadě to ale výsledek nijak neovlivňuje. Aplikace je napsaná v anglickém jazyce, neboť jde o jazyk, který je světově rozšířený a aplikaci tak může používat širší spektrum lidí.

Bohužel jsem byl hodně omezen časem, který jsem mohl práci věnovat, a tak se mi nepodařilo implementovat všechny své návrhy, které by má aplikace mohla nabízet. Aplikace by mohla mít možnost tvorby sdílených scénářů, což by usnadnilo tvorbu testů, kde u množiny testů existuje sled společných kroků. Tím by nevznikali zbytečné duplikace a údržba těchto testů by tak byla daleko jednodušší. Také bych rád rozšířil výběr funkcí, které lze provádět. Pro zatím jsem se však omezil jen na omezený počet akcí, které jsou však nejčastěji používané. Také bych rád umožnil uživateli, aby nemusel spouštět testy na vlastním zařízení, ale aby využil možnost, že spustí test na prostředí, které by bylo někde na serveru s aplikací pro spouštění testů. S tím by také souvisela i možnost streamování průběhu testů, aby uživatel mohl sledovat průběh. Ještě chybí dodělat možnost výběru prohlížeče, na kterém testy spouštět, možnost registrace vlastního serveru, na kterém by měl uži-

vatel možnost spouštět testy. Přehled o výsledcích testů by mohl být doplněn o přehledné grafy a jiné podstatné informace. Ceněnou možností by mohlo být i odesílaní výsledků na email. A jelikož se jedná o automatizované testy, měla by zde být i možnost plánování pravidelného spouštění testů, nebo také spuštění testů po té, co byla vydaná nová verze testované aplikace a to automaticky. Věřím, že možnosti, které by tato aplikace mohla nabízet, je další spousta.

Aplikaci je prozatím možné spouštět jen lokálně, ale rád bych, aby byla nasazena do provozu (na příklad na vlastním serveru, který zatím nevlastním). Případně si lze zaplatit i hostování.

Nicméně jsem použil aplikaci ve stávajícím stavu a jde o fungující celek. V aplikaci jsem si vytvořil několik testů, které jsem zkoušel spustit a tyto testy proběhly zcela úspěšně a to opakovatelně i v několika pokusech.

SEZNAM POUŽITÉ LITERATURY

1. Software Engineering | Prototyping Model. *Geeks for geeks.* [Online] https://www.geeksforgeeks.org/software-engineering-prototyping-model/.

2. Software Engineering | Incremental process model. *Geeks for geeks.* [Online] https://www.geeksforgeeks.org/software-engineering-incremental-process-model/.

3. **Ghahrai, Amir.** Incremental Model. *Testing Excellence.* [Online] 2. 12 2018. https://www.testingexcellence.com/incremental-model/.

4. **PAL, SAYAN KUMAR.** Software Engineering | Spiral Model. *Geeks for geeks.* [Online] https://www.geeksforgeeks.org/software-engineering-spiral-model/.

5. **Mgr. Jiří MARTINŮ, doc.Ing.Petr ČERMÁK, Ph.D.** *METODIKY VÝVOJE SOFTWARE.* Olomouc : Moravská vysoká škola Olomouc, o. p. s., 2018.

6. Software Engineering | Rapid application development model (RAD). *Geeks for geeks.* [Online] https://www.geeksforgeeks.org/software-engineering-rapid-application-development-model-rad/.

7. **Beck, Kent, a další, a další.** Manifest Agilního vývoje software. *Agile manifesto.* [Online] 2001. http://agilemanifesto.org.

8. **PAL, SAYAN KUMAR.** Software Engineering | Agile Development Models. *Geeks for geeks.* [Online] https://www.geeksforgeeks.org/software-engineering-agile-development-models/.

9. **ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ.** *Řízení kvality softwaru: průvodce testováním.* Brno : Computer Pres, 2013. ISBN 978-80-251-3816-8.

10. Black Box Testing. *Software Testing Fundamentals.* [Online] http://softwaretestingfundamentals.com/black-box-testing/.

11. White Box Testing. *Software Testing Fundamentals.* [Online] http://softwaretestingfundamentals.com/white-box-testing/.

12. **Hlava, Tomáš.** Testování bílé a černé skříňky. *Testování softwaru.* [Online] 20. 8 2011. http://testovanisoftwaru.cz/tag/white-box/.

13. **Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh, and Stephan**

Weber. Foundation Level Extension SyllabusAgile Tester. *ISTQB.* [Online] 2014. https://www.istqb.org/downloads/send/5-agile-tester-extension-documents/41-agile-tester-extension-syllabus.html.

14. **Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (project manager), Debra Friedenberg, Matthias Hamburg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, and Eshraka Zakaria.** Foundation Level Syllabus. *ISTQB.* [Online] 2018. https://www.istqb.org/downloads/send/51-ctfl2018/208-ctfl-2018-syllabus.html.

15. ISTQB – Test Levels. *Get Software Services.* [Online] https://www.getsoftwareservice.com/481/.

16. Unit Testing. *Software Testing Fundamentals.* [Online] http://softwaretestingfundamentals.com/unit-testing/.

17. **Hlava, Tomáš.** Fáze a úrovně provádění testů. *Testování softwaru.* [Online] 21. 8 2011. http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/faze-testu/.

18. What is Regression Testing? *Smartbear.* [Online] https://smartbear.com/learn/automated-testing/what-is-regression-testing/.

19. **SM, Rajkumar.** Automation Testing Vs Manual Testing | SoftwareTestingMaterial. *Software Testing Material.* [Online] 16. 3 2019. https://www.softwaretestingmaterial.com/automation-testing-vs-manual-testing/.

20. Test Automation Framework. *Techopedia.* [Online] https://www.techopedia.com/definition/30670/test-automation-framework.

21. **SM, Rajkumar.** Most Popular Test Automation Framework Interview Questions. *Software testing material.* [Online] 22. 4 2019. https://www.softwaretestingmaterial.com/test-automation-framework-interview-questions/.

22. **Rajkumar.** Types of Test Automation Frameworks | Software Testing Material. *Software testing material.* [Online] 2. 3 2019. https://www.softwaretestingmaterial.com/types-test-automation-frameworks/#What-is-a-framework.

23. Types of Automation Frameworks. *3Qi Labs.* [Online] 19. 9 2014. http://3qilabs.com/types-of-automation-frameworks/.

24. Behavior Driven Testing in Automated testing . *Katalon.* [Online] https://www.katalon.com/sa/behavior-driven-testing/.

25. **Kagathara, Mehul.** Automated Testing using BDT (Behavior Driven Testing). *Infostretch.* [Online] 13. 2 2013. https://www.infostretch.com/blog/automated-testing-using-bdt-behavior-driven-testing/.

26. **Brian.** Best Automation Testing Tools for 2019 (Top 10 reviews). *Medium.* [Online] 26. 11 2017. https://medium.com/@briananderson2209/best-automation-testing-tools-for-2018-top-10-reviews-8a4a19f664d2.

27. *SeleniumHQ.* [Online] https://www.seleniumhq.org.

28. Simplify API, Web, Mobile Automation Tests. *Katalon.* [Online] https://www.katalon.com/.

29. Unified Functional Testing (UFT). *Micro Focus.* [Online] https://www.microfocus.com/en-us/products/unified-functional-automated-testing/overview.

30. What is QTP/UFT Automation Testing Tool? *Guru99.* [Online] https://www.guru99.com/uft-qtp-automation-testing.html.

31. TestComplete. *SMARTBEAR.* [Online] https://smartbear.com/product/testcomplete/overview/.

32. SoapUI. [Online] https://www.soapui.org/.

33. *Robot Framework.* [Online] https://robotframework.org.

34. *Ranorex.* [Online] https://www.ranorex.com.

35. Top 20 Best Automation Testing Tools in 2019 (Comprehensive List). *Software Testing Help.* [Online] 25. 4 2019. https://www.softwaretestinghelp.com/top-20-automation-testing-tools/.

36. Volba mezi .NET Core a .NET Framework pro serverové aplikace. *Microsoft.* [Online] 19. June 2018. https://docs.microsoft.com/cs-cz/dotnet/standard/choosing-core-framework-server?toc=%2Faspnet%2Fcore%2Ftoc.json&bc=%2Faspnet%2Fcore%2Fbreadcrumb%2Ftoc.json&view=aspnetcore-2.2.

37. Informace o webových aplikacích. *Adobe.* [Online] https://helpx.adobe.com/cz/dreamweaver/using/web-applications.html.

38. **Čápka, David.** MVC architektura. *ITnetwork.cz.* [Online] https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor.

39. **Bernard, Borek.** Úvod do architektury MVC. *zdroják.cz.* [Online] 7. May 2009. https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/.

40. **Smith, Steve.** Přehled ASP.NET Core MVC. *Microsoft.* [Online] 8. January 2018. https://docs.microsoft.com/cs-cz/aspnet/core/mvc/overview?view=aspnetcore-2.2.

41. Dokumentace Entity Framework. *Microsoft.* [Online] https://docs.microsoft.com/cs-cz/ef/#pivot=entityfmwk&panel=entityfmwk1.

42. Code-Based Migration in Entity Framework 6. *Entity Framework Tutorial.* [Online] https://www.entityframeworktutorial.net/code-first/code-based-migration-in-code-first.aspx.

43. Seriál Začínáme s ASP.NET. *dotnetportal.cz.* [Online] https://www.dotnetportal.cz/clanky/serial/6/Zaciname-s-ASP-NET.

44. **SPAANJAARS, Imar.** *Beginning ASP.NET 4.5 in C# and VB.* [editor] 1. Indianapolis : Wrox;, 2013. ISBN 9781118311806.

45. **ŠOCHOVÁ, Zuzana a Eduard KUNCE.** *Agilní metody řízení projektů.* Brno : Computer Press, 2014. str. 175. ISBN 978-80-251-4194-6.

46. **MYSLÍN, Josef.** *Scrum: průvodce agilním vývojem softwaru.* 1. Brno : Computer Press, 2016. str. 167. ISBN 978-80-251-4650-7.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API Rozhraní pro programování aplikací (angl. Application Programming Interface)

ASP Active Server Pages

BDD Behavior Driven Development

BDT Behavior Driven Testing

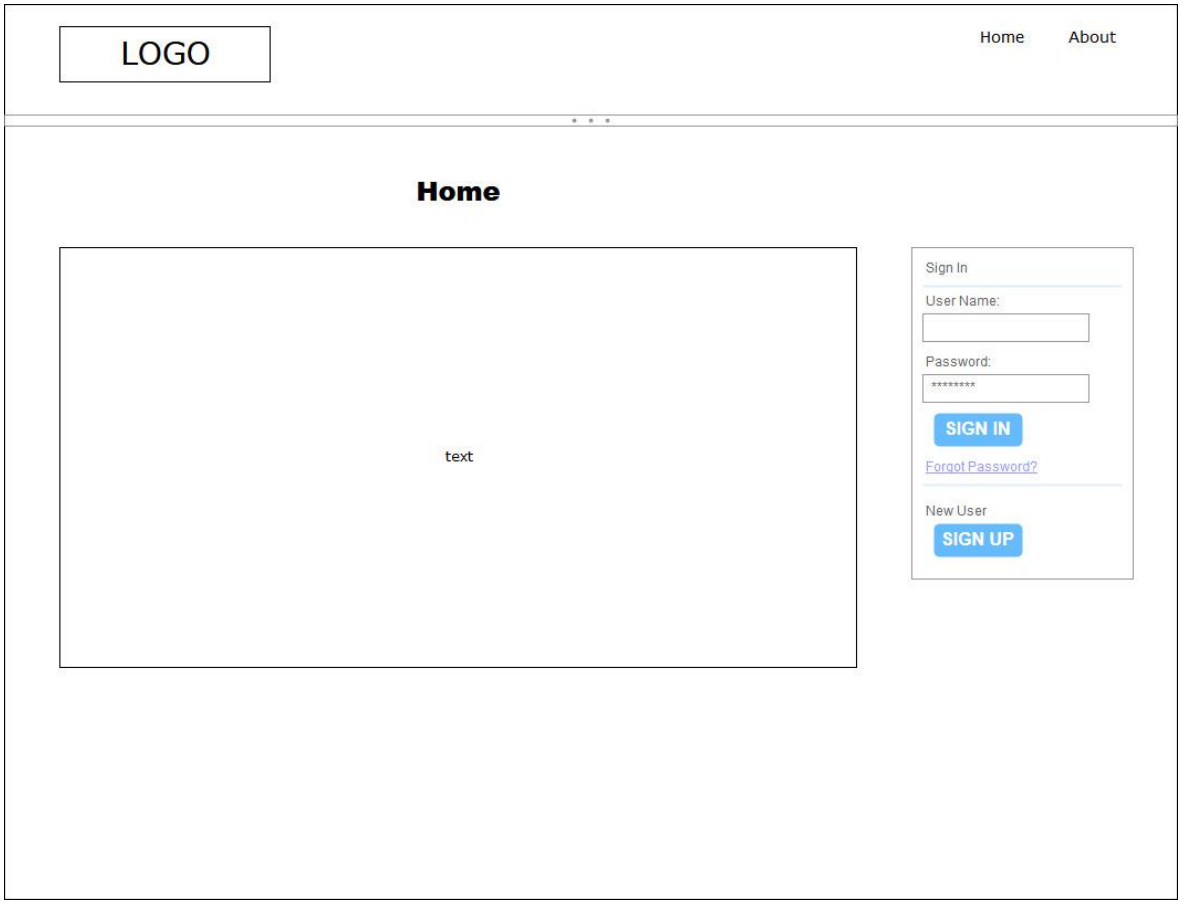
SW Software

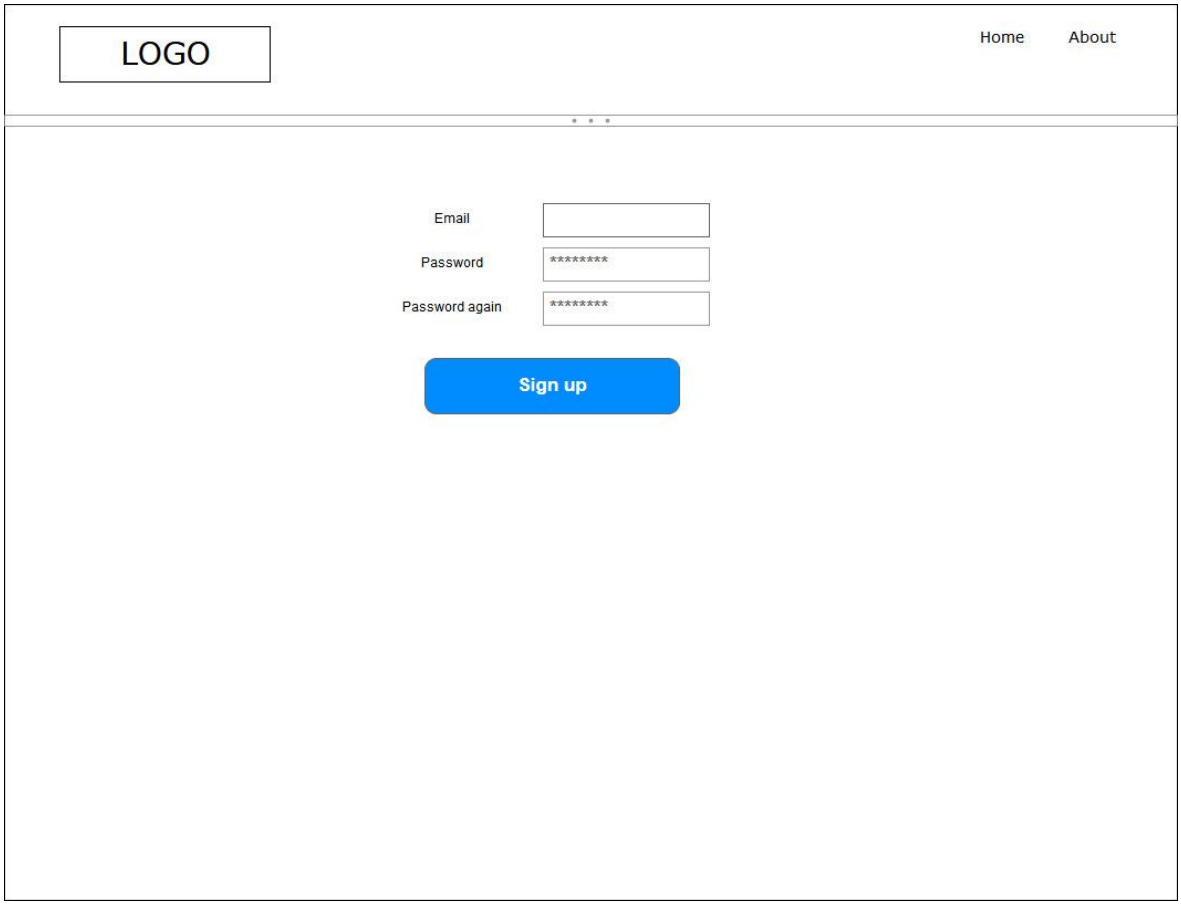
TDD Test Driven Development

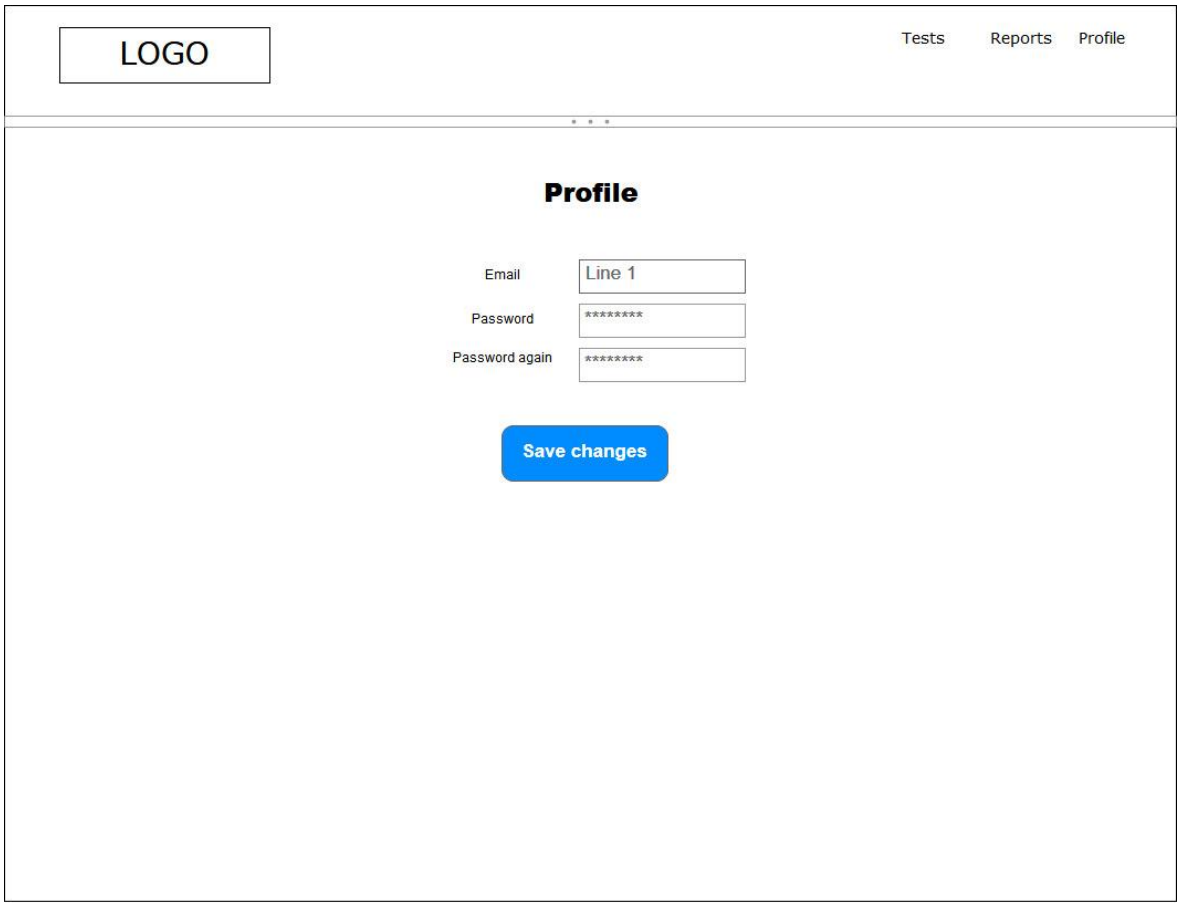
SEZNAM OBRÁZKŮ

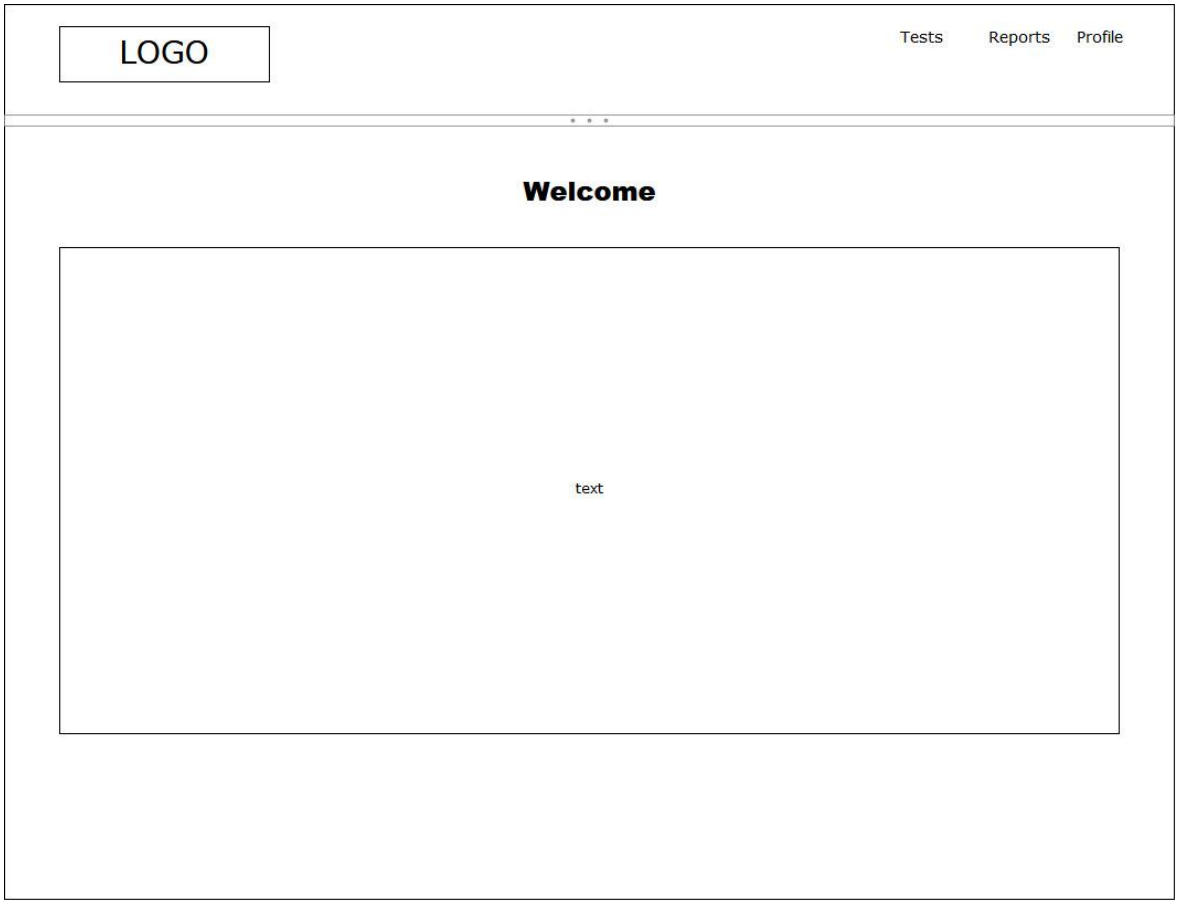
SEZNAM PŘÍLOH

PŘÍLOHA P I: NÁVRH DESIGNU




LOGO
Home
About
Email
Password
****
Password again
****
Sign up


LOGO
Tests
Reports
Profile
Profile
Email
Line 1
Password
****
Password again
****
Save changes


LOGO
Tests
Reports
Profile
Welcome
text

LOGO

Tests Reports Profile

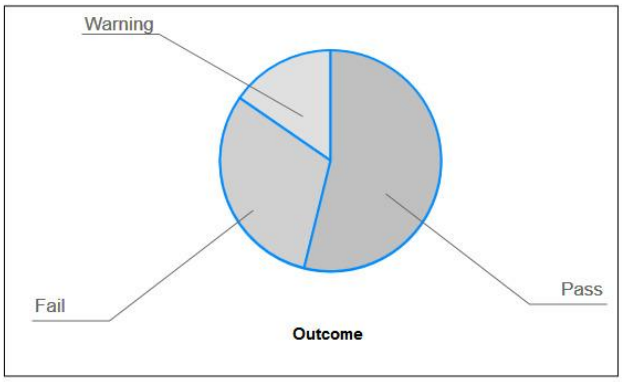
Reports

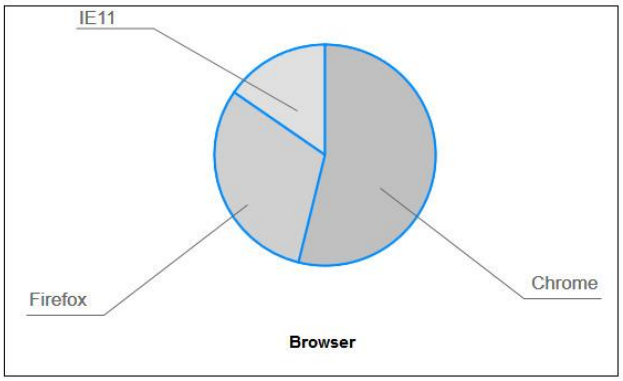
Last run

2018-10-16 15:42

Logs

Test name	Outcome	Browser
Test1	Passed	Chrome
Test2	Passed	Chrome
Test3	Passed	IE11
Test4	Failed	Chrome
Test5	Passed	Firefox
Test6	Passed	Chrome
Test7	Passed	Firefox
Test8	Passed	Chrome
Test9	Failed	Chrome
Test10	Passed	Firefox
Test11	Passed	Chrome
Test12	Warning	Chrome
Test13	Passed	Chrome
Test14	Failed	Chrome





LOGO

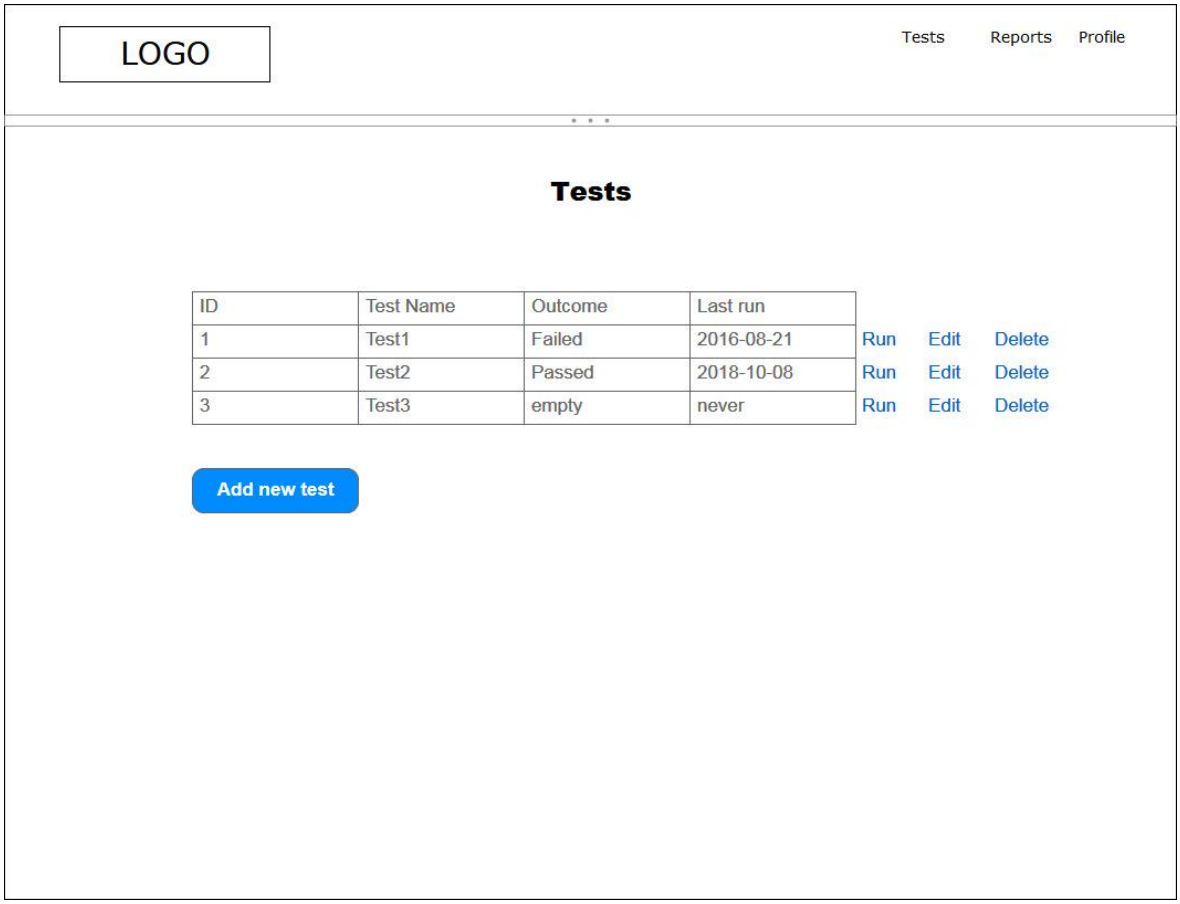
Tests Reports Profile

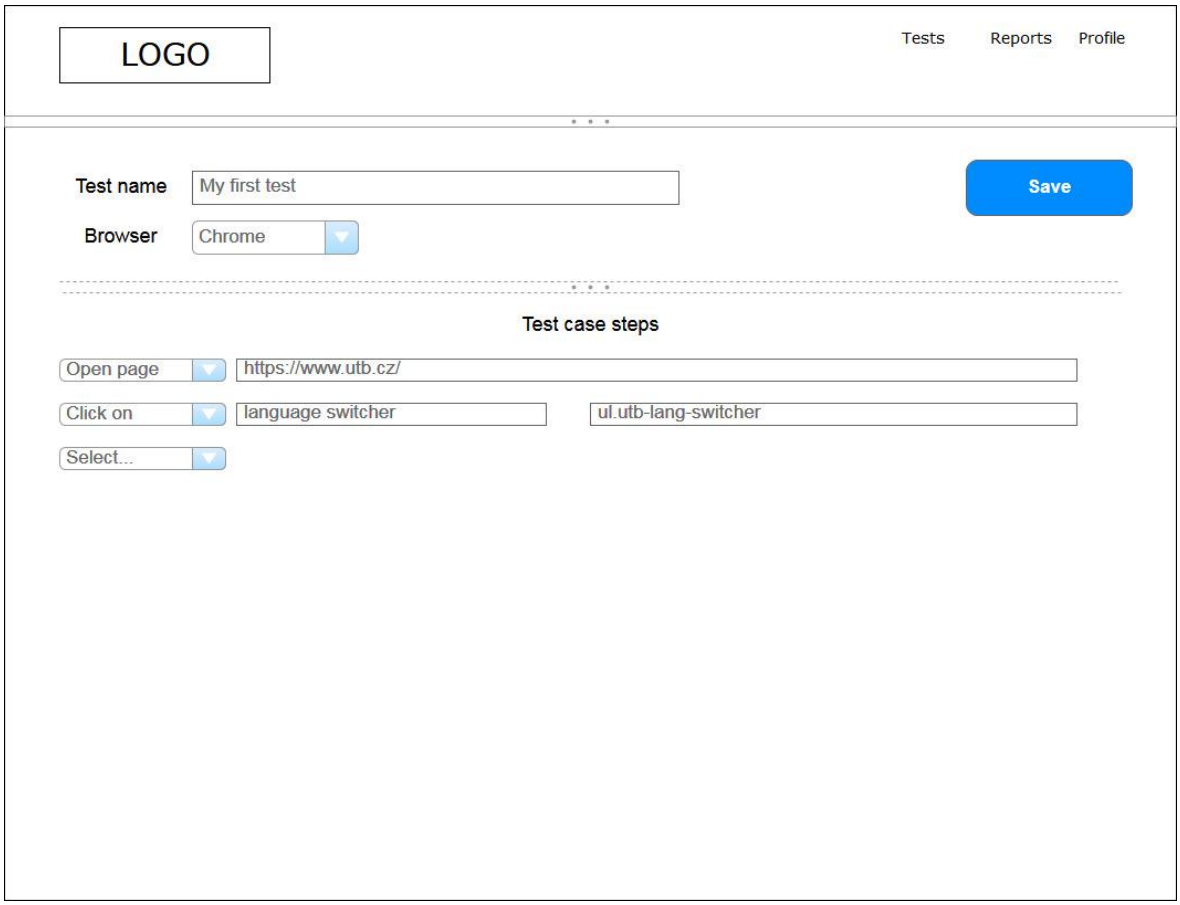
Test1 log

Started:	2018-10-17 17:32:16
Duration:	00:00:14
Browser:	Chrome
Outcome:	Pass

Steps:

Open page	*https://www.google.cz/*		
Write	*UTB*	to	*Find textbox*
Click on	*Find by Google*		
Assert text	*UTB: Univerzita Tomáše Bati ve Zlíně*	in	*First result*


LOGO
Tests
Reports
Profile
Tests
ID
Test Name
Outcome
Last run
1
Test1
Failed
2016-08-21
Run
Edit
Delete
2
Test2
Passed
2018-10-08
Run
Edit
Delete
3
Test3
empty
never
Run
Edit
Delete
Add new test


LOGO
Tests
Reports
Profile
Test name
My first test
Save
Browser
Chrome
Test case steps
Open page
https://www.utb.cz/
Click on
language switcher
ul.utb-lang-switcher
Select...