VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

INFORMAČNÍ SYSTÉM PRO SPRÁVU TESTŮ

TEST MANAGEMENT TOOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MIROSLAV NOVÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ALEŠ SMRČKA, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19500/2016/xnovak1k

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Novák Miroslav**
Obor: Informační technologie
Téma: **Informační systém pro správu testů**
Test Management Tool
Kategorie: Analýza a testování softwaru

Pokyny:

1. Analyzujte současný stav informačních systémů pro správu testů (test management tool).
2. Specifikujte požadavky na informační systém pro správu testů v rámci platformy Testos. Kladťe důraz na interoperabilitu systému s jinými systémy sloužící pro vývoj projektu (automatizované spouštění testů, verzovací systémy, reportování, správa dokumentace).
3. Imlementujte systém jako webovou službu a aplikaci.
4. Demonstrujte informační systém na netriviálním příkladu. Ověřte funkcionalitu webové služby s nadpolovičním pokrytím funkcí. Ověřte funkcionalitu webové aplikace s pokrytím všech klíčových částí.

Literatura:

- Davis, C.: Test management best practices. IBM, 2006. Dokument dostupný online: http://www.ibm.com/developerworks/rational/library/06/1107_davis/
- Domovská stránka projektu TestRail: http://www.gurock.com/testrail/
- Domovská stránka projektu Spira: https://www.inflectra.com/SpiraTest/

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrčka Aleš, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Cílem této práce je specifikovat, navrhnout a implementovat informační systém pro správu testů, který bude fungovat jednak samostatně a jednak v rámci platformy Testos. Zvolený problém byl vyřešen vytvořením klientské webové aplikace a implementací webové služby pomocí REST API, která zajišťuje interoperabilitu s jinými nástroji. Vytvořené řešení umožňuje řídit základní proces testování, který sestává ze správy projektů, správy testovacích požadavků, správy testovacích sad a případů a správy jednotlivých testovacích běhů.

Abstract

The aim of this thesis is to specify, design and implement a test management tool that will work both on its own and on the Testos platform. The problem was solved by creating a client web application and implementing a web service using the REST API, which ensures interoperability with other tools. The created system allows to manage the basic testing process, which consists of project management, test requirements management, test library management and management of individual testing runs.

Klíčová slova

Informační systém, testování softwaru, správa testů, PHP, Laravel, databáze, REST API

Keywords

Information system, software testing, test case management, PHP, Laravel, database, REST API

Citace

NOVÁK, Miroslav. *Informační systém pro správu testů*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

Informační systém pro správu testů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Aleše Smrčky, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Miroslav Novák
16. května 2017

Poděkování

Děkuji svému vedoucímu práce Alešovi Smrčkovi za jeho rady a konstruktivní připomínky, díky nimž se výsledek práce posunul o velký kus vpřed.

Obsah

Kapitola 1

Úvod

Testování je ve světě informačních technologií důležitá disciplína, díky které lze zajistit kvalitu produktu či služby. Přestože testování odjakživa souviselo s vývojem softwaru, v dnešní době jsou stále populárnější nové techniky řízení vývoje softwaru, v nichž hraje právě testování hlavní roli. Mluvíme zejména o přístupu DevOps[1] a její podmnožině Continuous integration[2].

S rozvojem testování přišla potřeba mít systém, který umožňuje spravovat celý proces testování a zároveň umožňuje spolupráci více testerů. Testovací proces se skládá z více částí. Především ze správy jednotlivých projektů, které mají být testovány, správy jejich požadavků a správy jednotlivých testovacích sad včetně testovacích případů. Dále se tento proces skládá ze spouštění a provádění testovacích běhů. Systém, který umožňuje řídit výše zmíněný proces, se označuje jako informační systém pro správu testů.

Na půdě Fakulty informačních technologií Vysokého učení v Brně vzniká v současnosti platforma Testos, která vyvíjí a zdržuje nové komplexní testovací nástroje. V rámci této platformy vznikla potřeba mít vlastní systém pro správu testů, jehož požadavky budou splňovat požadavky platformy. Přestože již podobné nástroje existují, tyto nástroje jsou víceúčelové. Systém, který byl v rámci bakalářské práce vytvořen, je jednoúčelový a lehce rozšiřitelný. Důležitým aspektem byla taktéž interoperabilita s dalšími nástroji platformy.

Hlavním přínosem této práce je analýza a specifikace nástroje pro správu testů, který splňuje požadavky platformy Testos. Dalším přínosem je návrh a implementace tohoto nástroje.

Struktura dokumentu

V kapitole 2 se nachází stručný úvod do problematiky testování softwaru včetně uvedení základní terminologie, která je dále v práci užívána. Kapitola 3 se zabývá analýzou existujících řešení. Dále tato kapitola obsahuje výčet a popis všech požadavků na systém pro správu testů, které se v průběhu řešení práce objevily. Kapitola 4 představuje jednotlivé technologie, které byly pro implementaci výsledného systému použity a jejich alternativy. V kapitole 5 se nachází návrh systému pro správu testů. Kapitola se zaměřuje na návrh testovacího procesu a databáze. Dále jsou zde uvedeny diagramy, které popisují architekturu systému. Kapitola 6 představuje vybrané implementační problémy, které v průběhu práce vznikly. Kapitola 7 představuje, jak byla ověřena funkcionalita výsledného systému.

[1] DevOps – https://en.wikipedia.org/wiki/DevOps
[2] Continuous integration – https://en.wikipedia.org/wiki/Continuous_integration

Kapitola 2

Stručný úvod do testování softwaru

> Program testing can be used to show the presence of bugs, but never to show their absence!
>
> Edsger W. Dijkstra

Pro správné specifikování systému, který bude pomáhat v řízení procesu testování, je nutné prvně tento proces definovat a pochopit.

2.1 Testování a jeho cíle

Cílem testování je ověřit, že program dělá to, co by měl a taktéž objevit defekty dříve, než je program nasazen do provozu. Proces testování má dva hlavní cíle [32]:

1. Demonstrovat vývojáři a zákazníkovi, že software splňuje dané požadavky. U softwaru vytvořeného na zakázku to znamená, že by měl existovat minimálně jeden test na každý požadavek. Pro generický[1] software to znamená, že by měly existovat testy pro každou funkcionalitu programu.

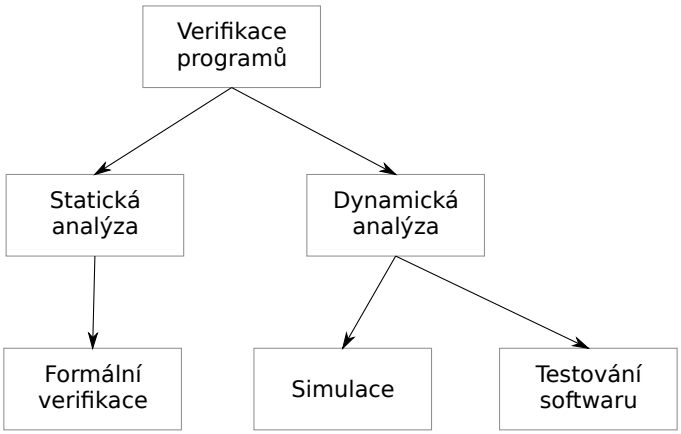
2. Objevit situace, v nichž je chování programu nesprávné, nežádoucí nebo nekorespondující s požadavky. Výše zmíněné jevy jsou důsledkem defektu.

Dále je důležité vložit testování do kontextu verifikace[2] softwaru. Jednotlivé typy verifikace lze nalézt na obrázku 2.1.

Přístupy k verifikaci softwaru jsou následující [31]:

- **Statická analýza**, která zkoumá vlastnosti softwaru bez jeho spouštění.
- **Dynamická zkoumá** ověřuje vlastnosti softwaru na základě provádění kódu.
- **Formální verifikace** s pomocí formálních metod ověřuje, že systém odpovídá specifikaci.
- **Testování** zkoumá software jeho spouštěním.

[1] Generický software – software, který se prodává libovolnému zákazníkovi, tj. krabicový software [26]
[2] Verifikace – ověření, zda software vyhovuje specifikaci [26]



Obrázek 2.1: Vztah verifikace programů. Zdroj: [31].

2.2 Testování z pohledu softwarového inženýrství

Testování nese v různých metodikách vývoje softwaru jinou roli a váhu. Tradiční lineární metodiky vývoje, například vodopádový model, definují následují posloupnost:

1. Analýza,
2. návrh,
3. implementace,
4. testování.

V-model, který je zobrazen na obrázku 2.2, navíc ke každé části vývoje definuje, jak by se měla daná část testovat.

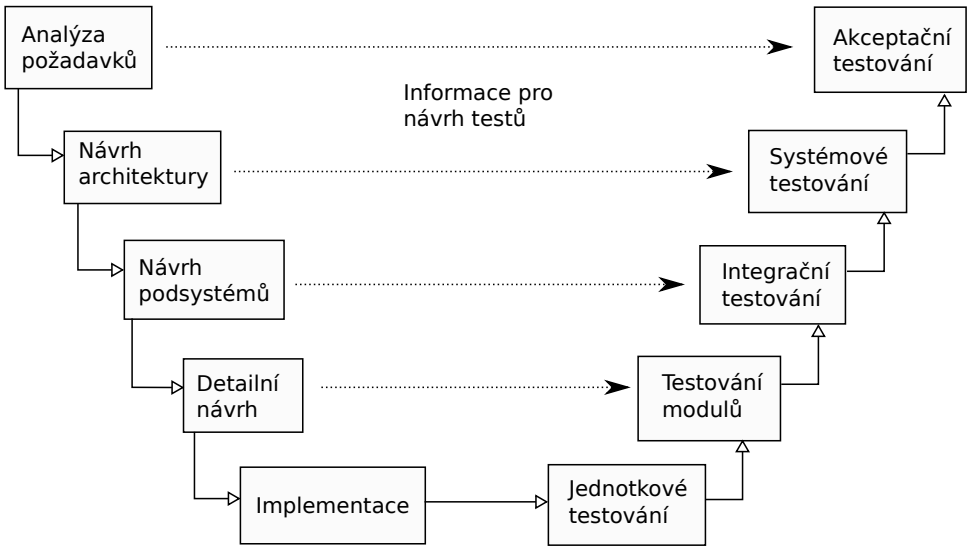
Tento lineární přístup může být v případech často se měnících požadavků nepružný. V případě změny požadavku je nutné znovu navrhnout, implementovat a otestovat systém. Proto začaly vznikat v 90. letech nové agilní metodiky. Tyto agilní metodiky upřednostňují inkrementální vývoj, který umožňuje dodat zákazníkům v krátké době nový produkt či funkcionalitu. Nové či změněné požadavky zákazníka lze poté zahrnout do další iterace vývoje [32]. Změna metodiky vývoje se dotkla taktéž testování. Testování v agilních metodikách neprobíhá až na konci vývojového procesu. Testování již probíhá současně s vývojem. Například metodika TDD[3] vyžaduje, aby před samotným vývojem funkcionality v podobě funkce či třídy byly vytvořeny testy.

Samostatnou kapitolou je v současnosti velmi oblíbený princip Continuous Integration. Tento princip spočívá v automatizovaném testování, které je spouštěno určitou událostí. Touto událostí může být například commit ve verzovacím nástroji Git.

2.3 Pojmy z oblasti testování používané v textu

Je nutné uvést a vysvětlit pojmy z oblasti testování, které se budou objevovat dále v textu. Tyto pojmy vychází z dokumentu [33] certifikační společnosti *International Software Testing Qualifications Board*.

[3] TDD – Test-Driven Development (vývoj řízený testy): https://en.wikipedia.org/wiki/Continuous_integration



Obrázek 2.2: V-Model. Zdroj: [17].

- **Defekt** je chyba v komponentě nebo systému, která způsobí nesplnění požadavků na software.
- **Proces** je posloupnost činností, které transformují vstupy na výstupy.
- **SUT** (system-under-test) je systém, který je právě testován.
- **Testovací požadavek** (angl. test requirement) je vlastnost či funkce komponenty nebo systému, která by měla být verifikovaná alespoň jedním testovacím případem.
- **Testovací případ** (angl. test case) je právě jeden test, který se skládá ze vstupních a očekávaných hodnot a taktéž z prefixových hodnot (tj. co se má vykonat před testem) a postfixových hodnot (tj. co se vykonat po testu).
- **Testovací sada** (angl. test suite) je množina testovacích případů.
- **Testovací běh** (angl. test run) je právě jedno spuštění určité množiny testovacích případů.
- **Množina testovacích běhů** (angl. test set). V názvosloví v dokumentu [33] je test set synonymem pro test suite. U jiných nástrojů pro správu testů (například *TestLink* či *PractiTest*) je však pojem test set používán pro označení množiny testovacích běhů. V této práci proto bude tímto pojmem myšlena právě množina testovacích běhů.

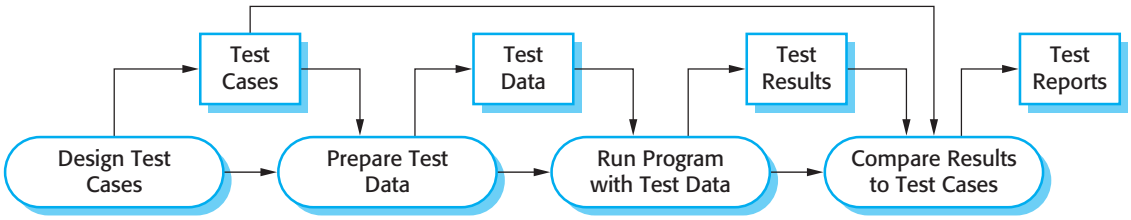
2.4 Popis klasického procesu testování

Testování se více podobá komplexnímu procesu než jedné aktivitě. Testovací proces určuje, co a v jakém pořadí má být uděláno. S pojmem proces testování souvisí pojem **testovací plán**. Norma ISO/IEC/IEEE 29119-3 pro dokumentaci testovacího procesu uvádí přesnou strukturu testovacího plánu. Dále tato norma uvádí, že smyslem testovacího plánu je [9]:

Definovat podrobný popis cílů, které mají být testováním dosaženy. Dále popis prostředků a časového harmonogramu, který vede k jejich dosažení.

Ian Sommerville uvádí tradiční proces testování (obrázek 2.3), který se skládá z:

1. Návrhu testovacích případů. Ty není možné zcela automatizovat.
2. Příprava testovacích dat. Testovací data jsou vstupy, které byly navrženy pro otestování systému. Testovací data je možné automatizovaně generovat.
3. Spouštění programu s testovacími daty, které lze automatizovat.
4. Porovnání skutečných výstupů s očekávanými výstupy. Porovnávání lze opět automatizovat.



Obrázek 2.3: Testovací proces. Zdroj: [32].

Kapitola 3

Analýza a specifikace požadavků na informační systém pro správu testů

Tato kapitola je rozdělena na tři části. První část obsahuje vymezení pojmu informační systém pro správu testů (3.1). Ve druhé části (3.2) se nachází analýza existujících systémů pro správu testů a kategorizace existujících řešení včetně detailní analýzy vybraného systému. Na základě této analýzy byl poté vytvořen seznam požadavků na systém, který se nachází ve třetí části kapitoly (3.3).

3.1 Informační systém pro správu testů

Pod správou testů (angl. test management) se dle odborné terminologie [33] rozumí: *Plánování, odhadování, monitorování a řízení testovacích aktivit, které obvykle vykonává vedoucí testovacího procesu.*

Informační systém pro správu testů (angl. test management tool) je poté [33]: *Nástroj, který poskytuje podporu pro správu testů a pro řízení části testovacího procesu. Často poskytuje více služeb. Například plánovaní testů, ukládání výsledků, sledování pokroku v rámci testování, řízení incidentů a reportování.*

3.2 Analýza existujících řešení

Na trhu existuje již celá řada systémů pro správu testů [1][2]. Jelikož systém, který je v rámci platformy Testos vytvořen, má částečně konkurovat těmto systémům, je vhodné tyto systémy analyzovat a případně nalézt pozitivní i negativní vlastnosti těchto řešení.

3.2.1 Klasifikace existujících řešení

Existující systémy pro správu testů lze klasifikovat do více kategorií. Dle **licence** používání lze dělit existující systémy na:

- **S otevřeným zdrojovým kódem** neboli Open Source. Obecně neexistuje na trhu mnoho systémů pro správu testů, které by uživatel mohl používat bezplatně a bez omezení [1]. Známým příkladem je *TestLink*, jehož komunita vývojářů je stále aktivní.

Dalším příkladem je *Tarantula*. Dle zdrojových kódů[1] však komunita vývojářů není již několik let aktivní.

- **Proprietární neplacené** jsou systémy, které uživatel může používat bezplatně, avšak zdrojové kódy jsou skryté a majetkem společnosti. Příkladem tohoto systému na správu testů je *XStudio* od společnosti *XQual*. Uživatel může verzi *XStudio Community* použít bezplatně, avšak oproti jiným verzím tohoto softwaru nese bezplatná verze mnohá omezení[2].

- **Proprietární placené** jsou nejpočetnější skupinou. O těchto systémech se taktéž hovoří jako o komerčních systémech. Tyto systémy standardně nabízejí bohatou sadu funkcionalit jako například vestavěnou integraci s dalšími nástroji (nejčastěji nástroje na evidenci chyb). Na základě analýzy bylo zjištěno, že tyto systémy se neprodávají jako krabicový software, ale jako služby s pravidelným měsíčním nebo ročním paušálem. Příkladem těchto komerčních produktů je *TestRail*, *Spira*, *qTest* a *PractiTest*. Zajímavým řešením je nástroj od společnosti *IBM*, který nese název *IBM Rational Quality Manager*.

Dle **formy aplikace** lze dělit existující systémy na:

- **Webové aplikace**, což jsou aplikace, které jsou poskytované uživatelům z webového serveru přes počítačovou síť internet nebo její vnitropodnikovou obdobou. Ke spuštění těchto aplikací potřebuje uživatel pouze internetový prohlížeč, proto jsou nezávislé na operačním systému uživatele [6].

  V současnou chvíli je naprostá většina systému pro správu testů distribuovaná jako webová aplikace. Dle způsobu **nasazení** lze tyto webové systémy dále dělit na:

  - **Cloudové služby** neboli SaaS[3]. Což jsou služby, které jsou spuštěny na webových serverech poskytovatele služby. Zákazník je tedy odstíněný od hardwaru, na němž služba běží. Poskytovatel taktéž ručí za běh služby [32]. Příkladem takového systému je *PractiTest* či *TestCaseLab*.
  - **Služba provozovaná u zákazníka** je služba, která běží na hardwaru uživatele nebo společnosti. Příkladem takového systému je *TestLink*.
  - **Rozšíření stávajícího systému** je další možností nasazení systému pro správu testů. Například aplikace *Zephyr* a *Xray* rozšiřují nástroj *JIRA* o možnosti správy testů a požadavků.

  Je nutné poznamenat, že některé aplikace umožňují jak provoz na serverech poskytovatele, tak na vlastním serveru. Poté záleží pouze na preferencích zákazníka jakou možnost si vybere. Těmito systémy jsou například *Test Collab* a *Spira*.

- **Desktopová aplikace** je nepříliš častá forma systému pro správu testů. Příkladem je aplikace *QABook*.

Dle přístupu k **evidenci chyb a problémů** lze dělit systémy na:

[1] *Tarantula* – zdrojový kód: https://github.com/prove/tarantula
[2] *XStudio* – licence: http://www.xqual.com/products/xstudio_license.html
[3] SaaS (Software as a Service) – Software jako služba

- **Systémy integrující existující řešení**. Tvůrci těchto systémů se rozhodli neimplementovat vlastní řešení, ale vytvořili možnost integrace s existujícími nástroji jako je *JIRA* či *Redmine*. Příkladem těchto systémů je *PractiTest* a *Testuff*.
- **Systémy obsahující vlastní řešení**. Jsou systémy, jehož autoři implementovali samostatné řešení, které nevyžaduje aplikace třetích stran. Příkladem je systém *Gemini* se svým řešením *Gemini Bug Tracking*.

3.2.2 Popis vybraného řešení s otevřeným zdrojovým kódem

Jako zástupce z Open Source řešení byl k představení vybrán *TestLink*, což je systém pro správu testů, který je implementovaný jako webová aplikace. Hlavními komponentami v systému *TestLink* jsou [20]:

- **Testovací projekt**, což je základní organizační jednotka v systému. Testovací projekt obsahuje testovací plán, testovací specifikaci, správu požadavků, inventář na správu hardwaru a specifické uživatelské práva.

  Jednotlivé projekty jsou zcela nezávislé a nesdílejí spolu žádná data.
- **Testovací specifikace** je struktura obsahující testovací sady a přidružené testovací případy. Každý projekt má právě jednu specifikaci.

  Testovací případy mají tzv. *Custom fields*, což jsou speciální atributy, které může administrátor projektu dodatečně konfigurovat. Testovací případy lze verzovat. Pakliže uživatel přiřazuje daný testovací případ do testovacího plánu, vznikají zde již ale omezení [20].
- **Testovací plán** reprezentuje spouštění a vyhodnocování testů z testovací specifikace. Testovací plán se skládá z jednotlivých množin testovacích běhů (angl. test set), které obsahují testovací případy.
- **Správa testovacích požadavků**, která umožňuje vytvářet a editovat požadavky. Navíc lze tyto požadavky pokrývat testovacími případy.

Velkou výhodou systému *TestLink* je podpora integrace s dalšími nástroji jako je *Bugzilla*, *MantisBT* a *JIRA*.

3.3 Požadavky na systém pro správu testů

Vývoj každého úspěšného a fungujícího systému začíná v analyzovaní a specifikování požadavků na tento systém. Požadavek na systém popisuje, co by systém měl dělat, jaké služby by měl poskytnout a v čem spočívá jeho omezení. Tyto požadavky reflektují potřeby zadavatele práce. Podle úrovně abstrakce se dělí požadavky na [32]:

- **Uživatelské požadavky**, což jsou tvrzení v přirozeném jazyce a diagramy popisující, co lze od jednotlivých služeb systému čekat.
- **Systémové požadavky** popisují, co vše musí být implementováno. Dokument, který obsahuje tyto požadavky, se nazývá Dokument Specifikace požadavků a jeho formát udává norma IEEE 830.

V rámci semestrálního projektu jsem vytvořil základní Dokument specifikace požadavků. Zde v bakalářské práci budou specifikovány uživatelské požadavky.

3.3.1 Funkční požadavky na systém

Následující seznam obsahuje výčet a popis funkčních[4] požadavků na systém pro správu testů. Tento seznam byl vytvořen na základě více zdrojů. Předním zdrojem bylo zadání bakalářské práce a dále pravidelné konzultace s vedoucím práce. Doplňujícím zdrojem je [14]. Jako v každém větším projektu se seznam požadavků v průběhu řešení práce stále rozrůstal. Proto bylo nutné na změny těchto požadavků dynamicky reagovat.

- **Správa SUT** (angl. SUT management). O SUT lze v tomto kontextu hovořit jako o projektech.

  Správa a řízení projektů je základní složkou softwarového inženýrství. V rámci systému pro správu testů není nutné implementovat komplexní systém na řízení projektů. Speciálních systémů na řízení projektů existuje již hodně [3] a v rámci systému na správu testů nyní stačí pouze základní funkcionalita. Jednotlivé požadavky:

  - Projekty lze vytvářet a poté i editovat.
  - Projekty se dělí na aktivní a archivované. Aktivní jsou ty, na nichž se stále pracuje (tj. stále je co testovat). Archivované projekty jsou projekty neaktivní. Aktivní i archivované projekty lze zobrazovat a prohlížet si jejich podrobnosti. Aktivní projekt lze na žádost archivovat.
  - Atributy, které je nutné u projektu uchovávat, jsou název projektu, popis projektu, popis postupu testování, odhadovaná doba ukončení a hardwarové a softwarové nároky na projekt.
  - Každý projekt má právě jednoho vlastníka. Vlastník může projektům přiřadit další uživatele, kteří k němu budou mít přístup.
  - Projekty lze vyhledávat a filtrovat podle jména a data vytvoření.

- **Správa testovacích požadavků** (angl. requirements management), což je proces dokumentace, analýzy, monitorování, prioritizování a potvrzování požadavků na software. Důležitou části správy požadavků je taktéž kontrola změn požadavku a propojování jednotlivých požadavků [13]. Obdobně jako u správy SUT, podobných systémů, které se soustředí pouze na správu požadavků, existuje již hodně.

  - Požadavky jdou vytvářet, editovat a archivovat. Požadavek je vždy pevně svázán s konkrétním SUT.
  - Nutné atributy u požadavků jsou jméno požadavku, popis požadavku a kritérium pokrytí.
  - Požadavky lze pokrývat jednotlivými testovacími případy. S pokrytím souvisí stav požadavku. Stav může být *Pokrytý* a *Nepokrytý*.
  - U požadavků jde sledovat jejich změny. Starší verze požadavku lze obnovit. Z obnovené verze požadavku lze tvořit verze nové.
  - Jednotlivé požadavky jde propojit s jinými. U daného požadavku lze poté zjistit po stisknutí příslušného tlačítka, s jakými dalšími požadavky tento požadavek souvisí.
  - Požadavky lze vyhledávat a filtrovat podle jména a stavu.

[4] Funkční požadavek - popisuje, co by systém měl umožňovat [32]

- **Správa testovací knihovny** (angl. test library management). Pod testovací knihovnou se rozumí testovací sady, které obsahují testovací případy. Testovací knihovna je nezávislá na projektu a mohou k ní přistoupit všichni uživatelé bez omezení.

  - Testovací sady jdou vytvářet, editovat a archivovat.
  - Nutné atributy u testovací sady jsou jméno sady, popis sady a cíle testovací sady.
  - K dané testovací sadě lze vytvářet testovací případy. Testovací případy jdou zadávat hromadně. Aplikace musí podporovat zrychlené vytváření testovacích případů. Zrychleným vytvářením testovacích případů se rozumí vytváření bez vložení dodatečných detailů.
  - Nutné atributy u testovacích případů vycházejí z definice testovacího případu. Viz 2.3. Je nutné si uchovávat jméno testovacího případu, popis, jednotlivé kroky, očekávaný výstup a taky prefixové a postfixové hodnoty.
  - Testovací případy lze editovat a archivovat.
  - Testovací případy lze vyhledávat a filtrovat dle jména a testovací sady.

- **Správa testovacích běhů** (angl. test set and run management). Testovací běhy jsou dále řazeny do množiny testovacích běhů.

  - Množinu testovacích běhů lze vytvářet. Při vytvoření je nutné definovat její jméno, popis a její přidružené testovací případy, ze kterých se testovací množina bude skládat.
  - Množina testovacích běhů vždy náleží jednomu SUT.
  - Po vytvoření množiny testovacích běhů lze editovat pouze její popis. Název množiny a přidružené testovací případy editovat nelze.
  - Testovací množinu jde archivovat.
  - Množinu testovacích běhů lze vyhledávat a filtrovat dle jména a stavu, tj. *Aktivní* a *Archivovaný*.
  - V detailu množiny testovacích běhů lze vytvářet nové testovací běhy. Testovací běh obsahuje všechny testovací případy, které obsahuje testovací množina. Testovací běh nemá jméno.
  - Testovací běh se nachází v různých stavech. Tyto stavy jsou: *Běžící*, *Dokončený* a *Archivovaný*. Tyto stavy lze libovolně měnit a lze pomocí nich testovací běhy vyhledávat a filtrovat.
  - U testovacích běhů jdou přehledně zobrazit jejich výsledky.

- **Provádění testů** (angl. test execution). Provádění testů je proces spouštění testů a porovnávání očekávaných a skutečných výstupů [5]. Spouštění je standardně přiřazeno na konkrétního testera nebo tým testerů. Provádění testů se dělí na:

  1. **Manuální testování**, které je charakteristické tím, že se tester dostává do role koncového uživatele a provádí předem definované činnosti (tj. testy) [4].

     Jednotlivé požadavky:

- Po otevření detailu testovacího běhu se zobrazí přehledná tabulka jednotlivých testovacích případů. Každý testovací případ je označen příslušnou barvou podle stavu, v němž se nachází. Existující stavy: *Netestováno* – žlutá, *Úspěch* – zelená, *Neúspěch* – červená, *Blokováno* – tmavě šedá. Z této tabulky lze stav testovacích případů měnit.
- Po rozkliknutí daného testovacího případu na přehledové tabulce je uživatel přesměrován na detail testovacího případu. Zde jsou uživateli zobrazeny všechny relevantní údaje k testovacímu případu (tj. popis, jednotlivé kroky, očekávaný výstup). Jde zde změnit stav případu a posunout se na další testovací případ.

2. **Automatizované testování**, které bývá někdy spojováno s pojmem regresní testování. Regresní testování je opakované provádění testů za účelem kontroly, zdali chyby odhalené v předchozích testech byly skutečně opraveny a jestli nebyly zavlečeny nové chyby. Protože těchto testů může být hodně, je vhodné je automatizovat [27].
   - V systému je nutná podpora pro automatizované spouštění testovacích běhů. Při každém spuštění se vytvoří nový testovací běh.
   - Systém zobrazuje výstup z automatizovaného spuštění. Ukazuje jaké množství testů z daného běhu bylo úspěšných a kolik neúspěšných. Dále zobrazuje dodatečné žurnálovací informace a údaje o tom, kolik času testování zabralo.

- **Monitorování výsledků testů** (angl. test results monitoring and dashboards).
  - Nástroj monitoruje nejdůležitější údaje z procesu testování. Na tyto potřeby se využívají tzv. dashboard. Dashboard umožňuje graficky prezentovat velké množství dat na malém prostoru a ve srozumitelné formě [22].

- **Správa týmů a uživatelů** (angl. team management and user management).
  - Aplikace umožňuje vytvářet týmy, editovat je a přidávat do nich uživatele.

- **Správa změn a verzování**
  - V systému je vše verzované a každá změna musí být zpětně dohledatelná. Každá komponenta (projekt, požadavek atd.) obsahuje správu verzí pomocí které lze prohlížet starší verze. Starší verze komponenty jde obnovit a vytvořit z nich novou verzi.

- **Integrace** (angl. integration). Pod integrací se rozumí schopnost systému interagovat s jinými externími nástroji, které slouží pro podporu vývoje a testování. Tím jsou myšleny nástroje na:
  - Evidenci chyb a problémů (např. *JIRA* či *Redmine*).
  - Správu verzí zdrojového kódu (např. *Github* či *Gitlab*).
  - Automatizované spouštění testů (např. *Jenkins* nebo *Bamboo*).
  - Správu dokumentace (*např. eFileCabinet*).

- **Import a export** testovacích požadavků a testovací knihovny.

3.3.2 Nefunkcionální požadavky na systém

Následující seznam je výčet nefunkcionálních požadavků na systém, které vychází ze zadání bakalářské práce. Další požadavky byly specifikovány po konzultaci s vedoucím práce.

- **Webová aplikace** je požadavek, aby byla aplikace poskytována pomocí webového serveru.
- **Webová služba** je požadavek, který stanovuje, že aplikace musí být přístupná a ovladatelná nejen přes rozhraní webové aplikace, ale taky přes veřejně dostupné API[5].
- **Rozšiřitelnost** neboli schopnost systému pružně reagovat na nové požadavky a případně upravovat již existující funkcionality.
- **Udržovatelnost zdrojových kódů** neboli psaní přehledného a čistého zdrojového kódu tak, aby mohl autor, případně jiný programátor, plynule navázat ve vývoji.
- **Dobře dokumentované implementační prostředí** je požadavek, který má za cíl, aby zvolené implementační prostředí bylo vhodně dokumentované a komunita okolo prostředí byla stále aktivní.
- **Responzivní design** je požadavek na to, aby byla výsledná aplikace dobře zobrazitelná na více typů zařízení. Příkladem mohou být mobily, notebooky a tablety.
- **Testovatelná implementace** je požadavek, aby existoval vhodný způsob, jak výslednou implementaci otestovat pomocí sady automatizovaných testů.
- **Podpora klávesových zkratek**.

3.3.3 Případy užití systému

Pro zachycení požadavků se často používají diagramy případů užití, které jsou součástí jazyka UML. Základní činností při vytváření diagramů je nalezení hlavních aktérů případů užití [26]. V případě tohoto systému jsou to tři hlavní účastníci: Vykonavatel testů, Standardní tester a Hlavní tester.

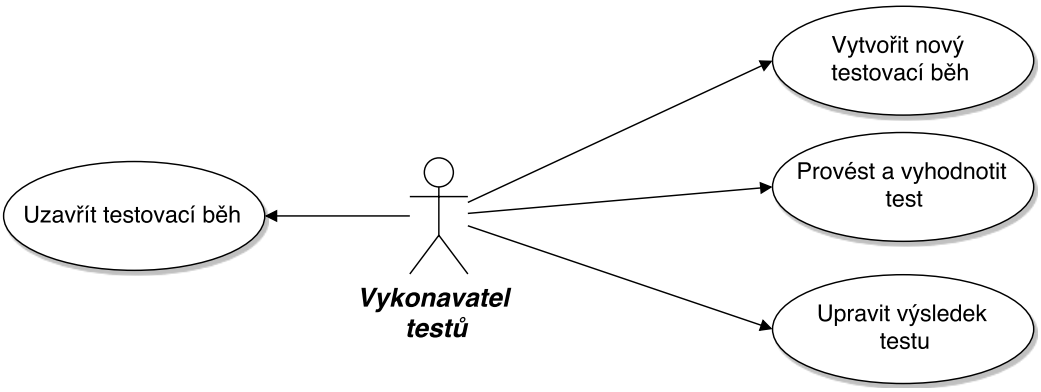
Vykonavatel testů

Vykonavatel testů (angl. test executor) je uživatel, jehož úkolem je spouštět jednotlivé testovací běhy, vykonávat je a vyhodnocovat výsledky. Viz obrázek 3.1. Tento uživatel provede kroky daného testu, splní prefixové hodnoty testu, zkontroluje, zdali očekávaný výstup koresponduje se skutečným výstupem a provede postfixové hodnoty testu.

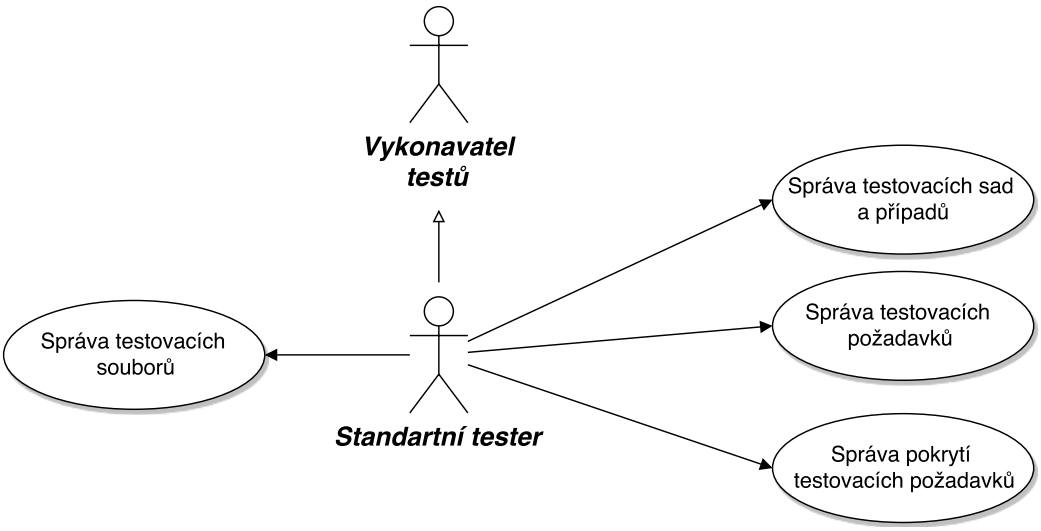
Standardní tester

Standardní tester je uživatel, který definuje jednotlivé testovací sady a případy. Tento uživatel dále vytváří množiny testovacích běhů a může spravovat testovací požadavky. Viz obrázek 3.2.

[5] API (Application Programming Interface) – rozhraní pro programování aplikací. Viz https://cs.wikipedia.org/wiki/API



Obrázek 3.1: Vykonavatel testů – diagram případu užití.



Obrázek 3.2: Standardní tester – diagram případu užití.

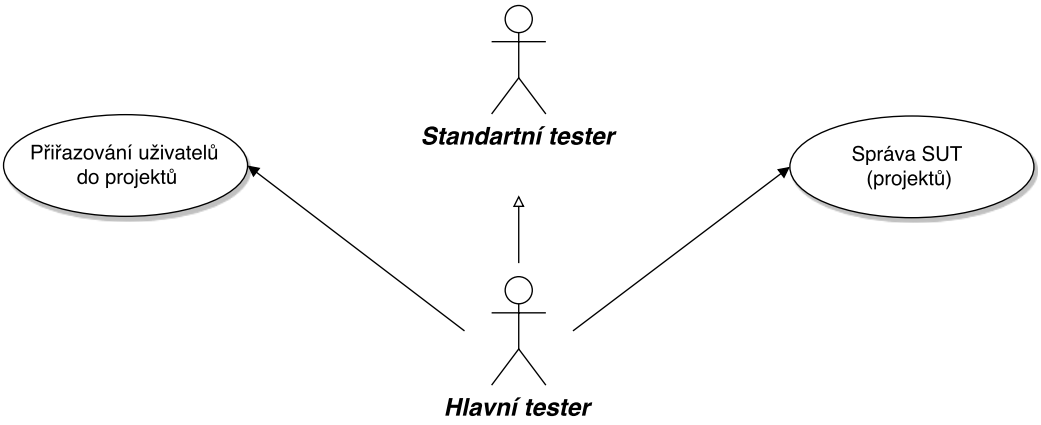
Hlavní tester

Hlavní (nebo také vedoucí) tester, je uživatel, který může vykonávat vše jako předešlí uživatelé. Navíc tento uživatel spravuje projekt a přiřazuje tomuto projektu jednotlivé uživatele.

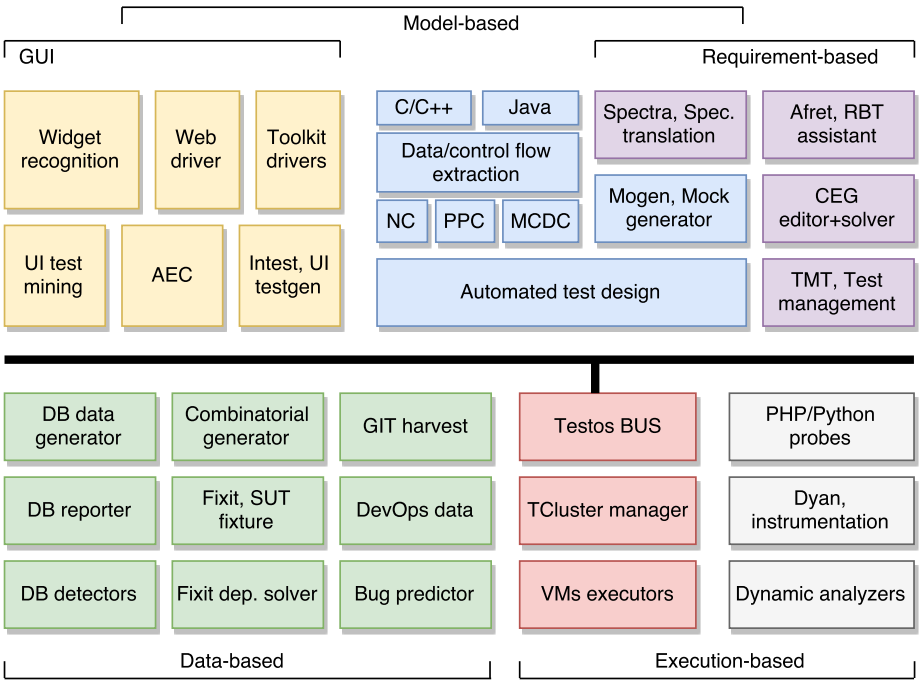
3.3.4 Platforma Testos

Jak bylo zmíněno v úvodu, informační systém vzniká pod platformou Testos (Test Tool Set) [15]. Testos je nový projekt, který vznikl na Fakultě informačních technologií Vysokého učení technického v Brně. Cílem projektu je vytvoření sady nástrojů podporující automatizované testování softwaru. Nástroje v platformě Testos (viz obrázek 3.4) kombinují různé úrovně testování a lze je řadit do několika kategorií: testování založené na modelech (Model-based), testování založené na požadavcích (Requirement-based), testování grafického uživatelského rozhraní (GUI), testování založené na datech (Data-based) a dynamická analýza (Execution-based).

Na vývoji platformy se podílí převážně studenti FIT. Ať již ve svém volném čase nebo ve formě diplomové práce. V aktuálním vývoji nástrojů pro testování založené na požadavcích jsou nástroje pro návrh testovacích případů ze specifikace požadavků [19], nástroj



Obrázek 3.3: Hlavní tester – diagram případu užití.



Obrázek 3.4: Dílčí části platformy Testos.

pro tvorbu CEG grafů a tvorbu testovacích případů založených na CEG [16] a poté nástroj pro správu testovacích požadavků a testů (tato bakalářská práce).

Role vyvíjeného systému v platformě Testos

Na obrázku 3.4 jsou zobrazeny jednotlivé komponenty, které tvoří platformu Testos. Například v modré části lze vidět extrakci grafu toku řízení a následné generování testů. V šedé části je naopak testování vícevláknových aplikací a dynamická analýza.

Jednotlivé komponenty platformy jsou připojeny na sběrnici, která je na obrázku nazvaná *Testos Bus*. Skrze tuto sběrnici je připojen i systém na správu testů. Tento systém pracuje aktivně, kdy volá jednotlivé komponenty, ale také i pasivně, kdy je volán jinými komponentami platformy.

Hlavní rolí vyvíjeného systému v platformě Testos není jen poskytnutí alternativního systému pro správu testů, ale systému, který jednotným způsobem umožní komunikovat s vybranými komponentami (ovládat je, přijímat a ukládat výsledky) a pomocí automatizace nahradit jinak ručně prováděné procesy správy testů.

Kapitola 4

Popis použitých technologií

V dnešní době existuje velmi mnoho technologií, které jsou vhodné pro vývoj webových aplikací a webových služeb. V následující kapitole jsou představeny technologie, které byly zvažovány a poté popis technologií, které byly nakonec pro implementaci použity. Samotný popis je rozdělen do tří částí.

V první části (4.1) je vysvětleno, co je to aplikační framework. Druhá část se zabývá technologiemi pro serverovou část (4.2), což je ta část aplikace, která zahrnuje ve třívrstvé architektuře [34] právě aplikační a datovou vrstvu. Serverová část obsahuje veškerou logiku aplikace a řídí přístup k databázi. Třetí část (4.3) představuje technologie pro implementaci klientské části, neboli technologie pro tvorbu webových uživatelských rozhraní. Ve třívrstvé architektuře náleží uživatelské rozhraní právě do prezentační vrstvy.

4.1 Aplikační framework

V souvislosti s použitými technologiemi je nutné definovat framework (nebo také aplikační framework či aplikační rámec). Douglas Schmidt definoval framework [30]:

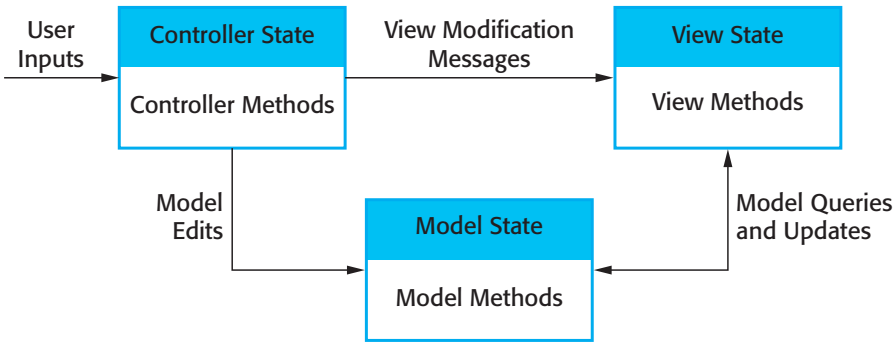
> *A framework is an integrated set of software artifacts (such as classes, objects, and components) that collaborate to provide a reusable architecture for family of related applications.*

Framework poskytuje podporu obecných vlastností, u kterých je velká pravděpodobnost, že budou použity v aplikacích, pro které framework slouží. Dále framework nabízí znovupoužitelnost díky speciálně navržené architektuře tříd. Znovupoužitelnosti je dosaženo hlavně pomocí dvou klíčových návrhových a architektonických vzorů: *MVC* a *Inversion of control* [32]. Při výběru vhodného frameworku pro vývoj webových aplikací je nutné sledovat, zdali tyto vzory podporuje.

4.1.1 Architektonický vzor MVC

Webové aplikační frameworky jsou v dnešní době založeny na architektuře Model-View-Controller (MVC), který je zobrazen na obrázku 4.1. Podstata této architektury spočívá v rozdělení aplikační logiky a uživatelského rozhraní. MVC se neoznačuje jako návrhový vzor, ale jako architekturální vzor. MVC se skládá ze tří komponent:

- **Model** (model) reprezentuje data a aplikační logiku nad těmito daty.



Obrázek 4.1: Architekturální vzor MVC. Zdroj: [32].

- **View** (pohled) reprezentuje vizualizaci dat, která jsou obsažena v modelu.
- **Controller** (řadič) přijímá a zpracovává požadavky od uživatele a aktualizuje model. Dále notifikuje pohled, že byl změněn model. Řadič tvoří prostředníka mezi modelem a pohledem.

4.2 Technologie pro implementaci serverové části

Následující podkapitola představuje dostupné technologie pro implementaci serverové části. Poté je představen vybraný framework a jeho funkcionality, které byly využity při implementaci výsledného systému.

4.2.1 Dostupné technologie

PHP

Je skriptovací programovací jazyk s dynamickou typovou kontrolou, který podporuje imperativní a objektově orientované programovací paradigma. Syntaxe jazyka je tzv. C-like, což znamená, že je v určitých aspektech (například řídící konstrukce) velmi podobná jazyku C. PHP je vyvíjeno jako Open Source [11].

Jazyk PHP byl navržen předně na programování webových aplikací. PHP je taktéž v současnosti zdaleka nejvíce používaným jazykem ve vývoji webových aplikací[1]. Uvažované aplikační frameworky:

- Nette Framework
- CakePHP
- Symfony
- Laravel
- Zend Framework
- Yii

[1] Usage of server-side programming languages for websites: https://w3techs.com/technologies/overview/programming_language/all

Python

Je skriptovací jazyk s dynamickou typovou kontrolou a podporou více paradigmat programování, který pochází z roku 1991. Python je vyvíjen jako Open Source. Syntax jazyka Python se oproti jiným jazykům liší, proto není C-like. Aktuálně jsou ve světě informačních technologií využívány dvě verze jazyka Python — *Python 2.7* a *Python 3.6* [12].

Python oproti jazyku PHP nebyl navržen přímo na vývoj webových aplikací. Přesto díky komunitě vývojářů existují i v jazyku Python frameworky, které jsou uzpůsobené na tvorbu webových aplikací. Uvažované aplikační frameworky:

- Django
- Flask

Java

Pod slovem Java je označován programovací jazyk Java a taky platforma pro běh aplikací. Jazyk Java je staticky typovaný a objektově orientovaný jazyk, který je od roku 2007 vyvíjen jako Open Source. Platforma Java je známá tím, že je multiplatformní. Co se týče překladu, jazyk Java se označuje jako hybridní jazyk. Prvně je zdrojový kód jazyka Java přeložen do bajtkódu, který je následně interpretován virtuálním strojem JVM [29].

Pro vývoj webových aplikací na platformě Java se používá její Enterprise edice – Java EE. Součástí platformy Java EE jsou například specifikace pro vývoj webových aplikací, webových služeb a přístupu k databázím. Vedle Java EE leží frameworky, které se snaží vývoj aplikací nad touto platformou ulehčit. Uvažované aplikační frameworky:

- Spring Framework
- Vaadin
- Play Framework

Další alternativy

Pro úplnost lze doplnit další alternativy pro vývoj serverové části aplikace jako programovací jazyk Ruby (s tím spojený framework Ruby on Rails), jazyk Javascript a s ním spojený Node.js. Poslední alternativou je platforma .NET od společnosti Microsoft.

4.2.2 Aplikační framework Laravel

Pro implementaci byl po zvážení vybrán aplikační framework **Laravel 5.3**, který je vyvinut v jazyku PHP 5. Důvod výběru byl především ten, že Laravel je nejpopulárnější a nejpoužívanější PHP framework současnost. Viz tabulka 4.1, která vznikla na základě dat ze serveru *Packagist*, což je hlavní repozitář pro balíčkovací nástroj Composer. Dalším důvodem byla aktivní komunita vývojářů kolem frameworku Laravel a taky osobní zájem o tento framework.

Popis frameworku Laravel

Laravel je mladý framework určený pro tvorbu webových aplikací, který vznikl v roce 2011. Jedná se o framework, který je možné využít zdarma a má otevřené zdrojové kódy. Lze ho používat pod licencí MIT. Aktuální verze frameworku je 5.4 a vyšla v lednu 2017.

Framework	Počet instalací za posledních 30 dní (k 22. 4. 2017)
Laravel	1 167 330
Symfony	786 011
Nette Framework	18 705
Yii	126 419
CakePHP	67 418

Tabulka 4.1: Počet instalací z Packagist.

Laravel vychází z existujících nástrojů. Využívá balíčky z frameworku Symfony a Doctrine. Co se týče architektury, Laravel využívá vzor MVC [10].

Popis funkcionalit frameworku Laravel využívaných ve vlastní implementaci

Následující výčet obsahuje krátký popis právě těch funkcionalit frameworku, které byly využity v implementaci výsledného systému. Laravel kromě toho nabízí další funkcionality jako například podpora ukládání do mezipaměti a odesílání elektronické pošty, které zatím nebyly v systému použity. Popis vychází z oficiální dokumentace k frameworku Laravel [8].

- **Routování** umožňuje spravovat všechny dotazy (HTTP požadavky) na aplikaci z jednoho souboru. V souboru *routes/web.php* lze zpracovat dotaz přímo, nebo lze delegovat dotaz na příslušný řadič. Ukázka roury:

  ```
  Route::get('requirements', 'RequirementsController@index');
  ```

  Roura výše deklaruje, že požadavek typu *GET /requirements* bude delegován do řadiče `RequirementsController` na metodu `index()`.

- **Autentizace** je funkce, která umožňuje vytvořit základní autentizaci aplikace zcela automatizovaně. Pod autentizací se rozumí v tomto kontextu registrace uživatelů a přihlašování/odhlašování uživatelů do aplikace.

  Laravel poskytuje tuhle funkcionalitu, jelikož téměř každý projekt potřebuje základní uživatelskou autentizaci. Programátor se tak může soustředit na jádro aplikace a nemusí ztrácet čas s vývojem autentizace, která je ve spoustě případů podobná. Laravel vytvoří nejen aplikační logiku za autentizací, ale i uživatelské rozhraní.

- **API autentizace** poskytuje základní autentizaci při programování API.

- **Migrace** je verzování změn v databázi, která umožňuje jednoduchou spolupráci s databázovými schématy napříč vývojářským týmem. Migrace řeší problém, když se vývojář rozhodne změnit schéma databáze. Migrací může distribuovat změny dalším vývojářům.

  Další výhoda spočívá v tom, že uživatel frameworku definuje databázi ve speciálních třídách frameworku Laravel. Po spuštění příkazu `php artisan migrate` poté *Laravel's schema builder* vytvoří databázi. *Scheme builder* si sám zjistí, která databáze je nastavena a vygeneruje odpovídající skript na vytvoření nebo aktualizování databáze.

- **Eloquent ORM** je objektově relační mapovaní. Jako každý jiný systém ORM[2] umožňuje i Eloquent ORM mapovat objekty na konkrétní dotazy v relační databázi. Z toho vyplívají následující výhody:
  - Odstínění od databázové vrstvy. S databází lze pracovat na úrovni aplikace a není nutné psát žádné SQL dotazy.
  - Nezávislost aplikace na konkrétním SŘBD[3]. Vytváření jednotlivých databázových dotazů zastává Eloquent, který se před vytvořením dotazu podívá do souboru *.env*, kde je definován SŘBD, pro něhož se má vygenerovat dotaz.
  - Při správném používání chrání před SQL injection pomocí tzv. *Parameter binding*.

  Příklad Eloquent ORM. Kód napsaný ve frameworku Laravel:

```php
<?php
$requirementsHistory = $requirementOverview
                            ->testRequirements()
                            ->orderBy('ActiveDateFrom', 'DESC')
                            ->get();
```

Kód 1: Příklad Eloquent ORM.

Je převeden na SQL dotaz:

```sql
select * from `TestRequirement`
where `TestRequirement`.`TestRequirementOverview_id` = ?
and `TestRequirement`.`TestRequirementOverview_id` is not null
order by `ActiveDateFrom` desc
```

Kód 2: Vygenerovaný SQL dotaz.

- **Middleware** poskytuje pohodlný mechanismus pro filtrování HTTP požadavků vstupujících do aplikace. Příkladem je middleware, který kontroluje, zdali je uživatel přihlášený do systému. Není-li uživatel přihlášený, je přesměrován na přihlašovací stránku. V opačném případě middleware nechá projít požadavek dále do aplikace.
- **CSFR ochrana** zajišťuje obranyschopnost před CSFR[4] útoky. Laravel automaticky generuje pro každou relaci přihlášeného uživatele CSRF token. Při požadavku typu POST je poté na straně serveru tento token ověřován.

[2] ORM – Object-relational mapping: https://en.wikipedia.org/wiki/Object-relational_mapping
[3] SŘBD – Systém řízení báze dat [34]
[4] CSFR – Cross-site request forgery: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

- **Validace** nabízí pohodlný způsob, jak validovat vstupní požadavky. Tato funkčnost se využívá například pro kontrolu vstupujících dat z formuláře aplikace. Ukázka:

```php
<?php
$this->validate($request, [
    'title' => 'required|unique:posts|max:255',
]);
```

Kód 3: Příklad validace ve frameworku Laravel.

  Výše zmíněný zápis zkontroluje, zdali položka se jménem *title* je v požadavku na aplikaci opravdu obsažena, zdali je unikátní v rámci databáze *posts* a jestli nepřesahuje 255 znaků.

- **Lokalizace** poskytuje podporu v případě, kdy chce uživatel frameworku vytvořit vícejazyčné aplikace.

- **Testování** je vlastnost, kterou se zabývá kapitola 7.1.1.

4.2.3 Databáze

Vzhledem k použití objektově relačního mapování ve frameworku Laravel, které nese název Eloquent ORM (viz 4.2.2), je možné nechat rozhodnutí o výběru konkrétního SŘBD až na dobu nasazení aplikace do produkce. Záležet bude na preferencích zadavatele práce. V současnou chvíli podporuje aplikace následující SŘBD [8]:

- MySQL
- Postgres
- SQLite
- SQL Server

Jelikož byla pro lokální vývoj vybrána databáze MySQL, mohlo být poté využito dodatečných nástrojů, které jsou spjaté s touto databází. Hovořit lze především o systému phpMyAdmin, který umožňuje jednoduchou správu MySQL databáze. Poté program MySQL Workbench, který byl využit pro návrh schématu databáze.

4.3 Technologie pro implementaci klientské části

Technologie pro tvorbu webových uživatelských rozhraní prošly za posledních několik let velikou změnou. V dnešní době jsou velmi populární technologie založené na jazyku Javascript, které umožňují psát takzvaně single-page[5] aplikace. Nejoblíbenější jsou právě javascriptové frameworky AngularJS a ReactJS.

V implementaci výsledného systému nejsou tyto pokročilé technologie pro tvorbu uživatelských rozhraní využity, jelikož se obsah implementace zaměřoval předně na serverovou část.

[5]Single-page appliaction: https://en.wikipedia.org/wiki/Single-page_application

4.3.1 HTML a CSS

HTML (HyperText Markup Language) je standardní značkovací jazyk, který je určený pro tvorbu WWW stránek a webových aplikací. První verze tohoto jazyka vznikla již v roce 1991. Tato verze, která byla známá pod označením HTML 0.9, už umožňovala rozčlenit text do několika logických úrovní, použít několik druhů zvýraznění textu a zařadit do textu odkazy a obrázky [25]. Aktuální verze jazyka HTML je HTML 5.0. Jednou z nejdůležitějších novinek nové verze jazyka je například vestavěná podpora přehrávání multimédií v prohlížeči a technologie *Canvas*, který slouží na interpretaci vektorové grafiky.

CSS, neboli kaskádové styly, slouží na definici vzhledu jednotlivých elementů HTML dokumentu. Pomocí CSS lze definovat například druh písma, způsob zarovnání textu, barvy pozadí jednotlivých elementů a mnohé další vlastnosti. Další výhoda použití kaskádových stylů spočívá v logickém oddělení obsahu (HTML) a vzhledu (CSS) do nezávislých souborů [25]. Aktuální verze jazyka CSS je CSS 3.

4.3.2 Javascript

S vývojem webových aplikací dnes již neodmyslitelně souvisí programovací jazyk Javascript. Javascript je skriptovací dynamicky typovaný jazyk. Javascript nabízí nezvyklou kombinaci vlastností. Jeho syntaxe vychází z jazyka C. Některá standardní rozhraní se podobají jazyku Java a objektový a funkcionální model je inspirovaný jazyky Scheme a Self. Tento jazyk nepoužívá třídy, jelikož je prototypově orientovaný [35].

Javascript slouží primárně na vývoj dynamických a interaktivních stránek, jelikož je interpretován přímo v prohlížeči.

4.3.3 Bootstrap

Bootstrap je nejvíce populární HTML, CSS a Javascipt framework, který velmi ulehčuje tvorbu responsivních webových aplikací. Bootstrap je distribuován jako sada *.css* a *.js* souborů. Dále jsou v této sadě obsaženy i soubory písem. Tyto soubory stačí pouze přiložit k projektu a poté využívat funkcionalit frameworku Bootstrap. V systému byl využit Bootstrap verze 3.

Bootstrap dále nabízí i několik základních šablon vzhledu webových aplikací, které lze využívat a libovolně upravovat. Pro účely vývoje uživatelského rozhraní byla využita Open Source šablona Startmin[6]. Její použití bylo spíše kontraproduktivní, jelikož bylo nutné tuto šablonu upravit pro účely bakalářské práce.

4.3.4 Použité knihovny jazyka Javacript

Zde je uveden seznam a krátký popis všech knihoven jazyka Javascript, které jsou použity ve výsledném systému pro správu testů.

- **jQuery** je knihovna, která výrazně rozšiřuje možnosti vývoje webových aplikací v jazyku Javascript. Tato knihovna je již implicitně importována, pakliže programátor využívá frameworku Bootstrap.
- **DataTables** je knihovna založená na jQuery, která výrazně rozšiřuje vlastnosti klasických HTML tabulek a dodává jim interaktivitu. Příkladem jsou možnosti řazení a filtrování řádků tabulky.

[6] Šablona Startmin: https://github.com/secondtruth/startmin

- **Multi-select** je taktéž knihovna postavená na jQuery, která poskytuje uživatelsky přívětivý formulářový element pro výběr více prvků.
- **AreYouSure** je knihovna založená na jQuery, která zobrazí uživateli varování v případě, kde editoval hodnoty ve formuláři a snaží se opustit aktuální stránku.
- **Stretchy** je knihovna, která slouží na automatickou změnu velikosti formulářového elementu v závislosti na obsahu elementu.
- **Highcharts**, což je knihovna, která slouží k vytváření interaktivních grafů.

Kapitola 5

Návrh informačního systému pro správu testů

V kapitole 3.3 byly rozebrány jednotlivé požadavky na aplikaci. Rozsah bakalářské práce však nemůže pokrýt všechny požadavky. Moje bakalářská se zaměřuje na správu projektů, správu testovacích sad a testovacích případů, správu testovacích požadavků, správu testovacích běhů a jejich spouštění.

5.1 Návrh procesu testovaní na základě požadavků

Proces, který byl použit ve výsledném nástroji, vychází právě z procesu, který byl představen v kapitole 2.4. Tento proces byl upraven tak, aby splňoval požadavky systému. Proces byl modelován pomocí behaviorálního UML diagramu, který se jmenuje diagram aktivit. Tento proces lze vidět na obrázku 5.1.

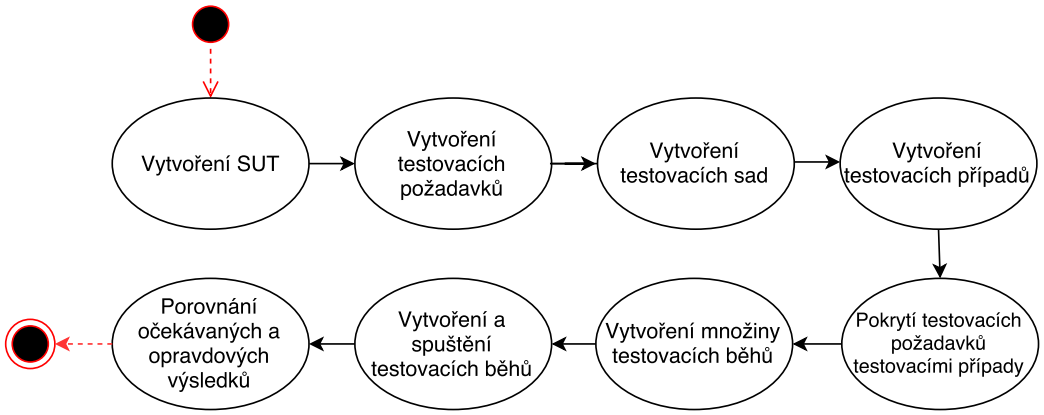
5.2 Návrh databáze

Databáze je esenciálním prvkem každého informačního systému. Díky databázím lze uchovávat stav systému, který je reprezentován hodnotami dat z databáze [21]. Pro účely systému bylo využito **relačních databází**, které vycházejí z teorie relační algebry. Standardním jazykem relačních databází pro vytváření databázových dotazů je jazyk SQL [34]. Existují však i jiné typy databází, které nejsou postaveny nad relační algebrou. Tyto typy databází se označují jako NoSQL databáze. Příkladem takové databáze je MongoDB[1], což je dokumentová databáze, která využívá serializační formát JSON.

5.2.1 Konceptuální modelování

Cílem konceptuálního modelování je analyzování požadavků na data, která budou uložena v databázi. Nejznámější a nejpoužívanější modelovací technikou konceptuálního modelování pro návrh relačních databází je entitně-vztahové modelování, jehož výsledkem je entitně-vztahový diagram nebo také ER diagram (angl. entity-relationship diagram) [34]. Výsledný ER diagram pro systém na správu testů lze nalézt v příloze na obrázku B.1.

[1] MongoDB – https://www.mongodb.com/



Obrázek 5.1: Navržený testovací proces systému pro správu testů.

5.2.2 Databázová podpora verzování změn

V průběhu řešení práce se objevil požadavek na verzování jednotlivých komponent systému (tj. testovacích požadavků a testovacích případů). Řešení tohoto problému nebylo složité. Problém byl ten, že bylo potřeba změnit chování téměř kompletního systému.

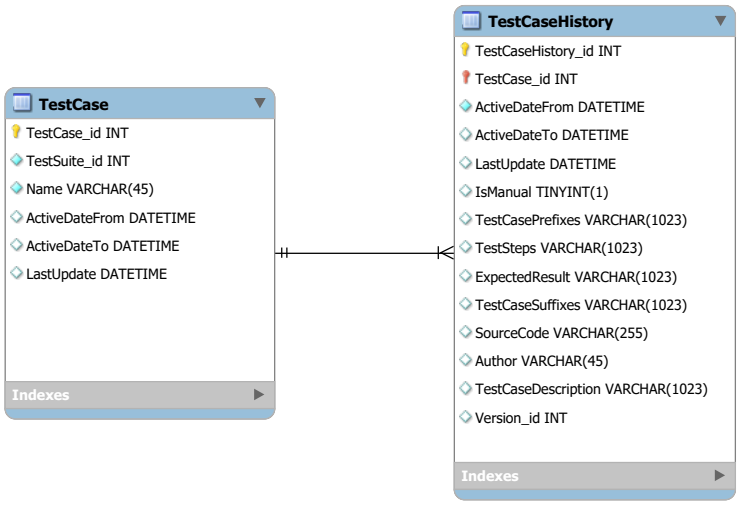
Pro implementaci tohoto požadavku bylo potřeba upravit databázi. V 1. kroku byly po vzoru temporálních databází přidány do jednotlivých záznamů časové razítka. V temporálních databázích se využívá časový model **valid-time**, což je interval, který udává, kdy jsou data v modelovaném světě validní [28]. Tato časová složka byla v databázi rozdělena do dvou atributů. Začátek doby platnosti je pojmenován `ActiveDateFrom` a konec doby platnosti je `ActiveDateTo`. Poté byl přidán atribut, který značí poslední úpravu nad daným záznamem. Tento sloupec tabulky se nazývá `LastUpdate`.

Ve 2. kroku bylo řešeno, jak změnit schéma databáze. Doktor Ralp Kimball pojmenoval tento problém *Slowly Changing Dimension* a definoval několik teoretických typů řešení [24]. Uvažováno bylo především nad těmito typy řešení:

- Přidání nového řádku tabulky, což je metoda, kdy se po každé aktualizaci záznamu v databázi přidá nový záznam do téže tabulky a záznam starý se deaktivuje.
- Přidání nové tabulky, která obsahuje historii změn. Při aktualizaci se vytvoří nový záznam do tabulky, která drží historii a starý záznam se deaktivuje.

Tento problém byl vyřešen právě přidáním nové tabulky. Důvodem bylo, že tímto způsobem vzniká přehlednější databáze a lépe dohledatelné změny. Například pro verzování testovacích případů byla vytvořena tabulka se jménem `TestCase`, která obsahuje název testovacího případu a identifikátor testovací sady. Poté byla vytvořena tabulka `TestCaseHistory`, která obsahuje identifikátor nadřazené tabulky `TestCase` a identifikátor verze. Viz obrázek 5.2.

Když si chce uživatel aplikace zobrazit aktuální verzi testovacího případu, provede se dotaz, který vrátí záznam z tabulky `TestCaseHistory`. Tento záznam má v atributu `ActiveDateTo` hodnotu null a v atributu `TestCase_id` identifikátor testovacího případu. V případě aktualizace záznamu je potřeba nalézt aktuální záznam, ten deaktivovat a vytvořit nový záznam v tabulce `TestCaseHistory`. Tomuto záznamu se inkrementuje číslo verze.



Obrázek 5.2: Databázové řešení verzovování testovacích případů.

5.3 Architektura systému

Architektonický návrh se zabývá návrhem celkové struktury systému. Výstupem návrhu architektury je popis, který říká, jak je systém organizovaný a jak spolu jednotlivé komponenty komunikují [32].

5.3.1 Architektura z pohledu závislostí a komunikace jednotlivých komponent

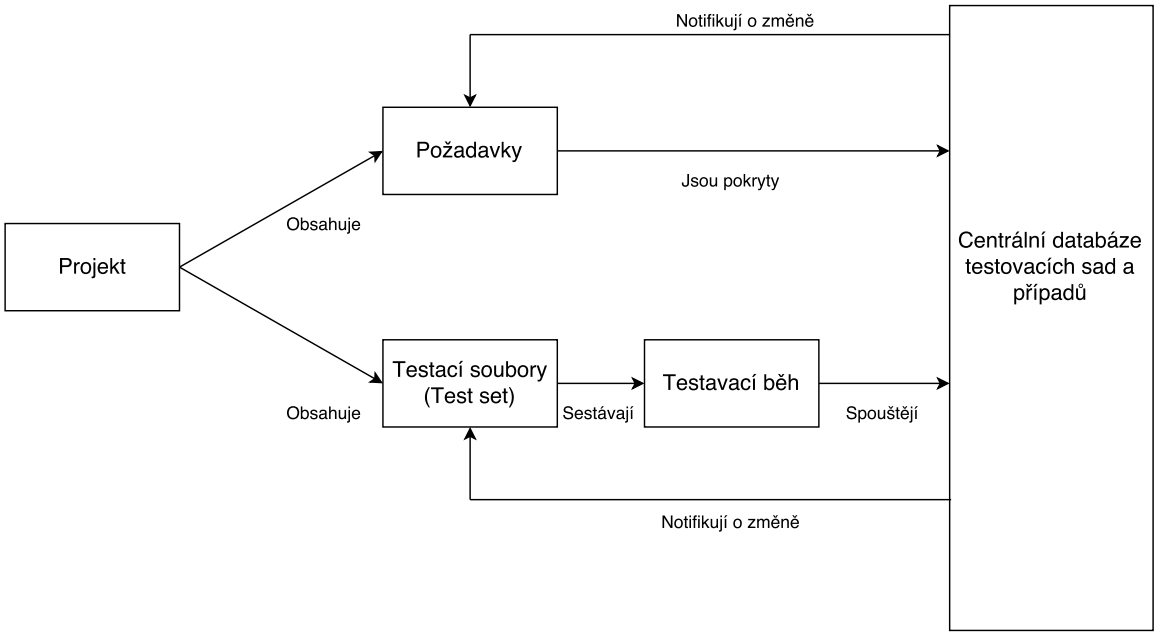
Na obrázku 5.3 lze vidět jednotlivé komponenty systému a jejich vzájemnou souvislost. Komponenty zde představují jednotlivé funkcionality systému specifikované v požadavcích. Například *Projekt* na diagramu reprezentuje správu projektů a *Požadavky* reprezentují správu požadavků.

5.3.2 Architektura z pohledu zpracování příchozího požadavku

Na obrázku 5.4 je znázorněna základní architektura webové aplikace. K jednotlivým komponentám, které zpracovávají příchozí požadavek, je na diagramu přidružen soubor, který za danou akci zodpovídá.

Zpracování příchozího požadavku ve frameworku Laravel lze popsat v následujících krocích:

1. Prohlížeč odesílá požadavek.
2. Soubor *routes/web.php* zpracuje příchozí požadavek a deleguje ho na příslušný řadič (angl. controller).
3. Před samotným spuštěním akce, která je definovaná v řadiči, se načtou služby, které bude řadič využívat. Tyto služby jsou definované v adresáři *app/Providers*. Poté požadavek prochází přes filtraci, kterou zajišťuje middleware. Jednotlivé middleware jsou definované v adresáři *app/Http/Middleware*. Viz. kapitola 4.2.2.



Obrázek 5.3: Komunikace a závislosti jednotlivých komponent systému. Projekt je zcela nezávislý na testovacích sadách a testovacích případech. Naopak požadavky a testovací běhy jsou definovány v rámci určitého projektu.

4. Řadič provádí akce s modelem. Může od něj získávat data, vkládat data, upravovat existující data či mazat data.

5. Po ukončení práce s modelem odesílá řadič požadavek na rendrování daného pohledu (angl. view).

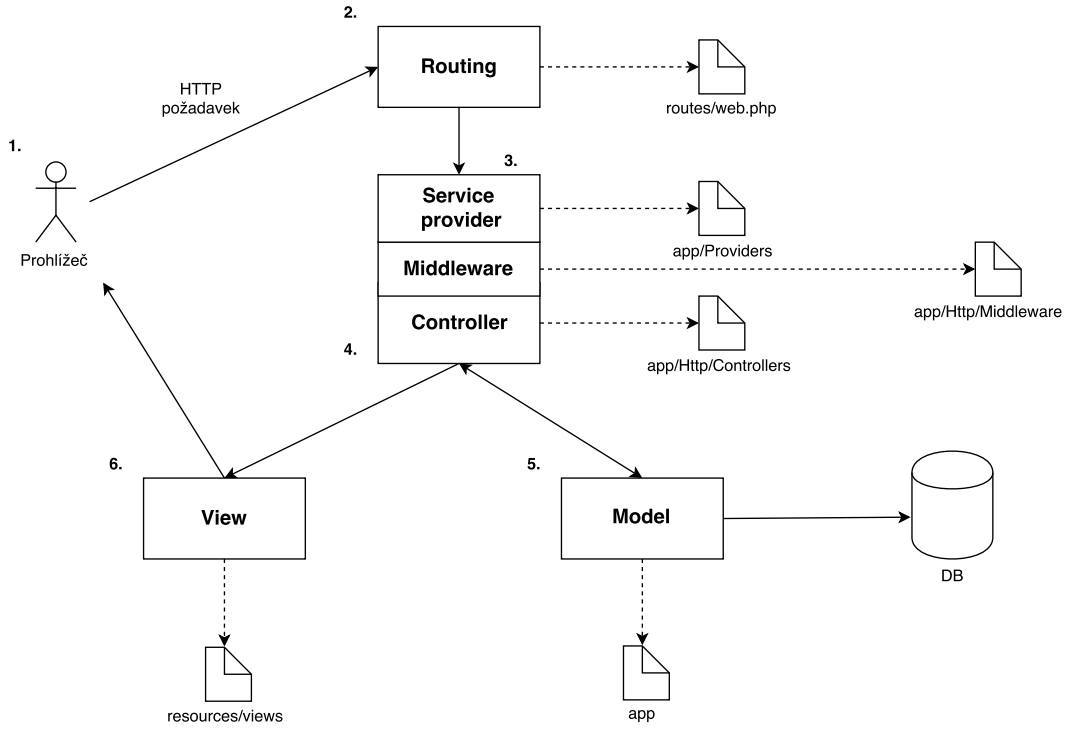
6. Po skončení render fáze je pohled odeslán zpět prohlížeči.

5.4 Návrh uživatelského rozhraní webové aplikace

Uživatelské rozhraní by mělo být v systému jednoduché a přehledné. To znamená, že v rozhraní nebudou prvky, které by uživatele rušily od práce. Koncept uživatelského rozhraní je zobrazen na obrázku 5.5.

Na horní liště se nachází logo platformy. Dále směrem vpravo je hlavní navigace, kde jsou odkazy na stránky, které pokrývají jednotlivé funkcionality jako správu testovacích požadavků nebo běhů. Dále napravo se nachází sloupec s vybraným projektem. Po rozkliknutí projektu se zobrazí nabídka, ve které si uživatel může zvolit jiný projekt. Nejvíce vpravo se nachází přihlášený uživatel. Po rozkliknutí se zobrazí detaily o uživatelském účtu a bude zde možnost odhlášení.

Tělo uživatelského rozhraní se skládá ze dvou částí. Z kontextuální nabídky, jejíž obsah záleží na dané stránce. Například v knihovně testů se v kontextuální nabídce zobrazují jednotlivé testovací sady. Kontextuální nabídka je vysouvací. V případě potřeby ji může uživatel skrýt tak, aby se mohl primárně soustředit na hlavní obsah. Hlavní obsah, jak již název napovídá, tvoří hlavní obsah dané stránky.



Obrázek 5.4: Architektura systému – pohled při zpracování příchozí požadavku.

5.5 Návrh webové služby

Webová služba nabízí způsob, jak si mohou dva stroje na síti vyměňovat informace. Použitím webových služeb mohou organizace poskytnout informace přístupné jiným programům přes definování a publikování webového rozhraní. Toto rozhraní definuje, jaké data jsou k dispozici a jak mohou být zpřístupněna [32]. Tomuto rozhraní se říká taktéž API. Na návrh a implementaci webových služeb se používají v dnešní době dvě technologie:

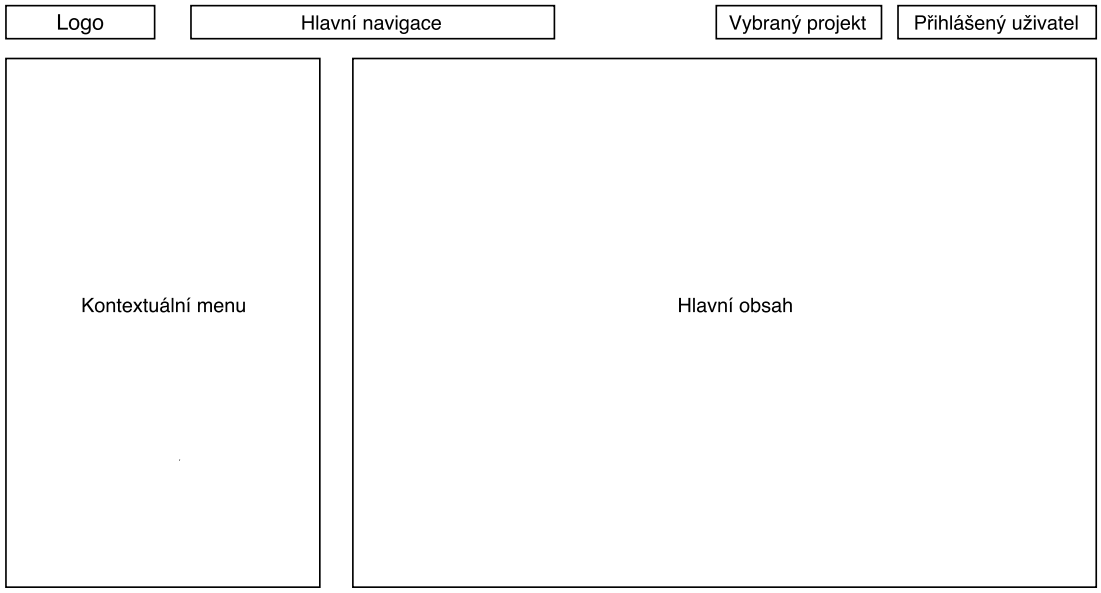
- **Protokoly SOAP a WSDL**, které jsou založené na serializačním formátu XML. SOAP je standart formátu zpráv posílaných mezi stroji. Naopak protokol WSDL je standart pro definici webového rozhraní.
- **RESTful webové služby**, které využívají principu architekturálního stylu REST. Standardně RESTful webové služby využívají protokol HTTP. Z HTTP metod se využívá GET, POST, PUT a DELETE. Jednotlivým URN[2], které následují za HTTP metodou, se říká zdroje (angl. resources). RESTful webové služby lze taky označit za REST API.

RESTful webové služby vznikly jako alternativa k těžkopádným webovým službám SOAP/WSDL. RESTful služby jsou tudíž mnohem jednoduší na implementaci [32]. Z toho důvodu bude systém pro správu testů využívat právě RESTful webové služby.

5.5.1 Návrh REST API

Dokumentaci k navrženému a implementovanému API lze nalézt v příloze E.

[2]URN – Uniform Resource Name: https://en.wikipedia.org/wiki/Uniform_Resource_Name



Obrázek 5.5: Návrh uživatelského rozhraní aplikace.

Návrh API vycházel z článku [7], který obsahuje doporučení jak API vytvářet. Dále je uveden seznam některých doporučení, které byla při návrhu brána v potaz:

- Zdroje by měly být pojmenovány jako podstatná jména.
- Zdroje by měly být pojmenovány v množném čísle.
- GET metoda by neměla měnit stav systému.
- Není-li je zdroj ve vztahu s jiným zdrojem, je vhodné použit tento zdroj jako zdroj dílčí. Například: `GET /cars/711/drivers/` vrátí seznam řidičů pro auto 711.
- API by mělo být vždy verzované.

Kapitola 6

Detaily z implementace informačního systému pro správu testů

V následující kapitole jsou představeny některé zajímavé detaily z implementace systému pro správu testů.

6.1 Implementace verzování změn

Existují 2 způsoby implementace navrženého řešení (viz kapitola 5.2.2) verzování změn:

1. **Řešení na databázové vrstvě**, které by pomocí databázových triggerů automaticky řešilo vzniklé situace.
2. **Řešení na aplikační vrstvě**, tedy přímo v kódu aplikace definovat požadované chování.

První řešení přináší výhodu odstínění problému od aplikační vrstvy. Dále toto řešení eliminuje chybu programátora, který by zapomněl vyvinout tuto funkčnost. Jelikož by měla být ponechána nezávislost aplikace na SŘBD, bylo zvoleno druhé řešení.

Show 10 entries Search:

id	Test case name	Version id	Test Suite
1	Not selected project	3	Create and edit project
2	Create project	2	Create and edit project
3	Create project without name	1	Create and edit project
4	Check if overview contains project	1	Create and edit project

Showing 1 to 4 of 4 entries Previous 1 Next

Obrázek 6.1: Ukázka verzování a upozornění na novou verzi testovacího případu.

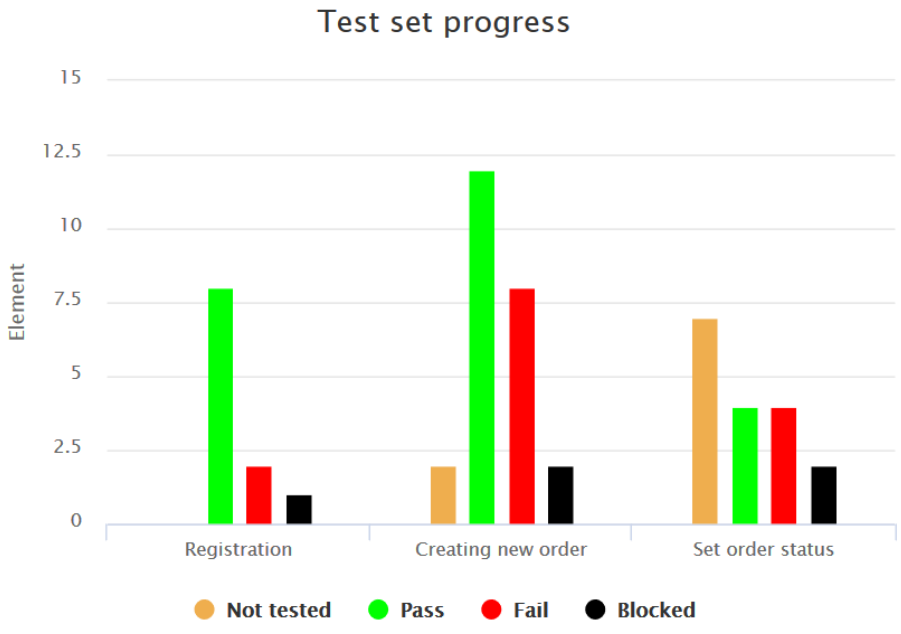
Dále byla implementována notifikace na změnu verze testovacího případu. V případě aktualizace testovacího případu, který zároveň pokrývá testovací požadavek, není možné automaticky aktualizovat i tento požadavek. Tento požadavek musí být pevně svázán s verzí testovacího případu, s níž byl původně pokryt. Zároveň je vhodné, aby byl uživatel s neaktuálností testovacího případu obeznámen. Ukázka z hotové aplikace je zobrazena na obrázku 6.1

Podobný případ nastává i při smazání testovacího případu. Opět není možné při smazání testovacího případu smazat i připnutý případ u testovacího požadavku. Problém je řešen obdobně. Nadřazená tabulka `TestCase` je deaktivována. V přehledu testovacích případů už nelze tento smazaný případ vidět. Ovšem v detailu testovacího požadavku tento případ stále vidět lze. Jen je označen jako smazaný.

6.2 Implementace grafů

Do finální implementace byly začleněny některé grafy, aby byl splněn požadavek na průběžné monitorování testovacího procesu. Tyto grafy jsou vykreslovány na úvodní stránce, která je nazvaná *Dashboards*. Implementovány byly celkem 3 grafy:

1. Výsečový graf, který zobrazuje poměr pokrytých a nepokrytých testovacích požadavků testovacím případem.
2. Sloupcový graf zobrazující testovací požadavky a počet pokrytých testovacích případů.
3. Sloupcový graf, který ukazuje vývoj jednotlivých množin testovacích běhů.



Obrázek 6.2: Příklad sloupcového grafu vývoje množin testovacích běhů.

Z výše uvedených grafů je právě 3. graf nejzajímavější. Skrze tento graf se uživatel hned dozví: v jakém stádiu jsou jednotlivé množiny testovacích běhů, kolik testů bylo úspěšných, kolik neúspěšných, kolik blokovaných a taktéž kolik ještě nebylo testováno.

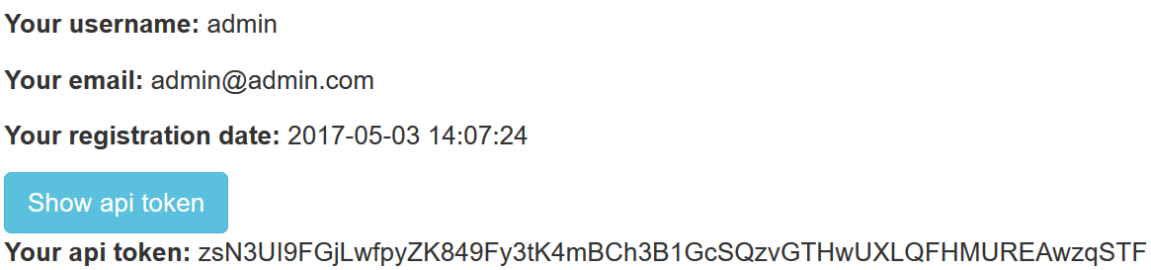
Implementace probíhala tak, že bylo nejdříve zjištěno pro každou aktivní množinu testovacích běhů, který testovací běh je nejvíce relevantní. Nejvíce relevantní běh je právě ten, který je ve stavu *Dokončený* a který byl zároveň naposled aktualizován. Z těchto běhů byly následně získány všechny obsažené testovací případy, kterými se iterovalo, čímž byla získána statistika o počtech jednotlivých stavů. Z těchto hodnot byl poté vytvořen graf. Viz obrázek 6.2.

6.3 Autentizace webové služby

RESTful webová služba musí být bezstavová. To znamená, že nelze použít pro autentizaci cookies nebo relaci (angl. session). Místo toho je nutné, aby byly přihlašovací údaje posílány při každém dotazu.

Při registraci nového uživatele je uživateli vygenerován tzv. API token, který uživatel posílá při každém HTTP požadavku do aplikace. Tento API token sestává z náhodně vygenerované sekvence 60 alfanumerických znaků. Více informací k REST API lze nalézt v příloze E.

Uživatel tento vygenerovaný API token nalezne po přihlášení do webové aplikace na stránce *User Profile* po rozklinutí tlačítka *Show api token*. Tato akce je zachycena na obrázku 6.3.



Obrázek 6.3: Ukázka detailu uživatele. Po kliknutí na tlačítko *Show api token* se uživateli zobrazí jeho API token.

Kapitola 7

Testování a nasazení výsledného systému

Testování systému je rozděleno na dvě části: testování serverové části a testování klientské části neboli uživatelského rozhraní. Mike Cohn ve své knize [18] tvrdí, že automatizovaného testovaní uživatelského rozhraní by měl tester dělat co nejméně. Nejvíce by mělo být vytvořeno automatizovaných jednotkových testů, poté testů služeb (pod čímž je myšlena aplikační vrstva nad uživatelským rozhraním) a až na vrcholu pyramidy se nachází automatizované testy uživatelského rozhraní. Právě z tohoto důvodu se pro serverovou část aplikace vytvořila automatizovaná sadu testů. Uživatelské rozhraní bylo otestováno ručně.

7.1 Akceptační testování serverové části systému

Aplikační framework Laravel, který byl použit pro implementaci, byl navržen tak, aby podporoval testování. V následující části se nachází popis testovaní pomocí frameworku Laravel a popis jednotlivých testovacích sad.

7.1.1 Testování pomocí frameworku Laraval

Ve frameworku Laravel je vestavěn testovací framework PHPUnit. Tento framework je již přednastaven, proto není potřeba žádné dodatečné konfigurace [8]. Při spuštění testů Laravel automaticky nastaví testovací prostředí webové aplikace. Důsledkem toho je, že žádné relace (angl. session) a vyrovnávací paměti se v průběhu testování neukládají.

Laravel poskytuje aplikační rozhraní pro vytváření a zasílání HTTP požadavků do aplikace. Toto rozhraní umožňuje například vyplňovat formuláře nebo kontrolovat HTTP odpovědi. Díky tomuto rozhraní lze jednoduše interagovat s aplikací bez závislosti na uživatelském rozhraní. Pomocí frameworku Laravel lze testovat i aktuální hodnoty v relaci. Dále Laravel nabízí funkčnost pro testování API. V neposlední řadě lze kontrolovat obsah databáze.

Ukázka testu ve frameworku Laravel

Máme test, který je zobrazen v kódu 4. Na 2. řádku je deklarováno, že se bude k aplikaci přistupovat přes předem vytvořeného uživatele *user*. Kroky tohoto testu jsou následující:

1. Navštívit stránku, která leží pod URN */library*.

```php
<?php
        $this->actingAs($user)
             ->visit('/library')
             ->click('#newSuite')
             ->seePageIs('/library/testsuite/create')
             ->type('TestSuite1', 'name')
             ->type('someDescription', 'description')
             ->type('someGoals', 'goals')
             ->press('submit');

        $this->seeInDatabase('TestSuite', [
                'Name' => 'TestSuite1',
                'TestSuiteDocumentation' => 'someDescription',
                'TestSuiteGoals' => 'someGoals'
            ]);
```

Kód 4: Příklad testu pomocí frameworku Laravel.

2. Kliknout na odkaz, který má ve své HTML značce identifikátor `newSuite`.
3. Zkontrolovat, zdali aplikace přesměrovala uživatele na správnou stránku.
4. Vyplnit a odeslat formulář, který slouží k vytvoření nové testovací sady.
5. Zkontrolovat, zdali databáze obsahuje novou testovací sadu.

7.1.2 Testovací sada pro webovou aplikaci

V rámci automatizovaného testování vytvořené webové aplikace bylo vytvořeno 5 testovacích sad, které jsou v následující části představeny. V popisu testů není obsažena fáze tzv. setup, která se spouští před každým testem. V této fázi se vytvoří databáze a uživatel, skrze něho se přistupuje do systému.

Registrace a přihlášení

Viz tabulka 7.1.

Vytvoření a úprava testovacích sad a testovacích případů

Viz tabulka 7.2.

Vytvoření a úprava projektu

Viz tabulka 7.3.

Testovací scénář	Očekávaný výsledek	Stav
Přesměrování na přihlašovací stránku, pakliže je uživatel nepřihlášen	Uživatel je přesměrován na přihlašovací stránku	Prošel
Registrace nového uživatele	Uživatel byl registrován	Prošel
Registrace uživatele se stejným emailem	Uživatel nebyl registrován	Prošel
Přihlášení a odhlášení uživatele	Uživatel byl přihlášen a poté odhlášen	Prošel

Tabulka 7.1: Testovací sada Registrace a přihlášení.

Testovací scénář	Očekávaný výsledek	Stav
Vytvoření nové testovací sady	Testovací sada byla vytvořena	Prošel
Vytvoření nové testovací sady bez zadání jména	Testovací sada nebyla vytvořena	Prošel
Přehledová tabulka testovacích sad obsahuje nově vytvořenou sadu	Přehledová tabulka obsahuje novou sadu	Prošel
Úprava již vytvořeného testovacího případu	Testovací sada byla upravena	Prošel
Vytvoření testovacího případu ve výše vytvořené sadě	Testovací případ byl vytvořen	Prošel
Vytvoření testovacího případu bez zadaného jména	Testovací případ nebyl vytvořen	Prošel
Detail testovací sady obsahuje testovací případ	Detail testovací sady obsahuje testovací případ	Prošel
Úprava testovacího případu – změna popisu a očekávaného výsledku	Testovací případ byl upraven	Prošel

Tabulka 7.2: Testovací sada Vytvoření a úprava testovacích sad a testovacích případů.

Testovací scénář	Očekávaný výsledek	Stav
V případě, kdy neexistuje žadný projekt – žádný projekt by neměl být vybrán	Není zvolen žádný projekt	Prošel
Vytvoření prvního projektu	Projekt byl vytvořen a zároveň zvolen jako vybraný	Prošel
Vytvoření projektu bez zadání jména	Projekt nebyl vytvořen	Prošel
Přehledová tabulka existujících projektů obsahuje projekt vytvořený výše	Přehledová tabulka obsahuje vytvořený projekt	Prošel
Úprava již vytvořeného projektu – změna popisu a popisu způsobu testování	Projekt byl upraven	Prošel

Tabulka 7.3: Testovací sada Vytvoření a úprava projektu.

Vytvoření a úprava testovacích požadavků

Viz tabulka 7.4.

Testovací scénář	Očekávaný výsledek	Stav
Vytvoření testovacího požadavku	Testovací požadavek byl vytvořen	Prošel
Vytvoření testovacího požadavku bez zadání jména	Testovací požadavek nebyl vytvořen	Prošel
Přehledová tabulka testovacích požadavků obsahuje nově vytvořený testovací požadavek	Přehledová tabulka obsahuje testovací požadavek	Prošel
Úprava testovacího požadavku – změna popisu požadavku	Testovací požadavek byl upraven	Prošel
Pokrytí testovacího požadavku testovacím případem	Testovací požadavek byl pokryt	Prošel

Tabulka 7.4: Testovací sada Vytvoření a úprava testovacích požadavků.

Vytvoření a spuštění testovacích běhů

Viz tabulka 7.5.

7.1.3 Testovací sada pro webovou službu

Pro testování webové služby byly taky vytvořeny automatizované sady testů. Každá sada pokrývá právě jeden dosažitelný zdroj přes navržené API. Každý test ze sady pokrývá právě jednu HTTP metodu, přes niž lze komunikovat s jednotlivými zdroji.

7.2 Akceptační testování klientské části systému

Testování uživatelského rozhraní probíhalo ručně. Testovací sada byla navržena tak, aby pokryla celý testovací proces (tj. proces, který podporuje vyvíjeny nastroj). Otestováním celého testovacího procesu se pokryje základní a nejdůležitější funkčnost webové aplikace. Testovací sada je k nalezení v tabulce 7.6.

7.3 Instalace a nasazení systému

Existují 3 způsoby jak instalovat a nasadit výsledný systém:

1. **Manuálně** nainstalovat všechny požadavky na aplikaci frameworku Laravel do lokálního PC. Seznam požadavků se nachází v oficiální dokumentaci [8]. Tento seznam sestává z PHP a také některých rozšíření jazyka PHP. Dále je nutné nainstalovat Composer a libovolnou databázi, která je podporovaná.

2. Použitím virtuálního stroje **Laravel Homestead**, což je předpřipravený Vagrant box, který poskytuje všechny služby, které jsou nutné pro běh aplikace.

3. Použitím **Laradock**, což je sada předpřipravených a předkonfigurovaných Docker obrazů, které poskytují prostředí pro běh a vývoj aplikací postavených na frameworku Laravel.

V příloze C se nachází instalační návod, který využívá právě Laradock. V příloze D je návod, jak spustit aplikaci skrze přiložený virtuální stroj.

Testovací scénář	Očekávaný výsledek	Stav
Vytvoření nového projektu	Projekt byl vytvořen a zároveň zvolen jako vybraný	Prošel
Vytvoření nového testovacího požadavku	Požadavek byl vytvořen. Je zobrazeno upozornění, že tento požadavek není pokrytý testovacím případem	Prošel
Vytvoření nové testovací sady	Testovací sada byla vytvořena. Je zobrazeno upozornění, že tento požadavek není pokrytý testovacím případem	Prošel
Vytvoření nového testovacího případu	Testovací případ vytvořen	Prošel
Pokrytí testovacího požadavku testovacím případem	Testovací požadavek je pokrytý testovacím případem. Je zobrazeno, že testovací požadavek je pokryt právě jedním testem	Prošel
Vytvoření množiny testovacích běhů a výběr konkrétního testovacího případu	Množina testovacích běhů byla vytvořen	Prošel
Vytvoření nového testovacího běhu	Testovací běh byl vytvořen	Prošel
Změna stavu testovacího případu v detailu testovacího běhu na "Prošlo"(angl. Pass)	Testovací běh byl vytvořen	Prošel
Změna stavu testovacího běhu v detailu testovací množiny na "Dokončeno"(angl. Finished)	Testovací běh byl vytvořen	Prošel
Kontrola, zdali stránka dashboards obsahuje správně vykreslené daty se správnými daty	Stránka dashboards obsahuje jeden výsečový graf a dva sloupcové	Prošel

Tabulka 7.6: Testování uživatelského rozhraní.

Testovací scénář	Očekávaný výsledek	Stav
Vytvoření množiny testovacích běhů a její naplnění testovacími případy	Množina testovacích běhů byla vytvořena	Prošel
Vytvoření množiny testovacích běhů	Množina testovacích běhů byla vytvořena	Prošel
Vytvoření množiny testovacích běhů bez zadání jména	Množina testovacích běhů nebyla vytvořena	Prošel
Vytvoření množiny testovacích běhů bez zadání testovacích případů	Množina testovacích běhů nebyla vytvořena	Prošel
Přehledová tabulka množin testovacích běhů obsahuje vyše vytvořenou množinu	Přehledová tabulka obsahuje množinu	Prošel
Úprava testovací množiny – změna jejího popisu	Množina testovacích běhů byla upravena	Prošel
Vytvoření testovacího běhu v množině testovacích běhů	Testovací běh je vytvořen	Prošel
Přehledová tabulka testovacích běhů v množině obsahuje nově vytvořený běh	Přehledová tabulka obsahuje běh	Prošel
Detail testovacího běhu obsahuje testovací případy definované v množině	Testovací běh obsahuje testovací případy	Prošel
Změna stavu testovacího případu v přehledové tabulce testovacího běhu	Stav testovacího případu byl změněn	Prošel
Změna stavu testovacího případu v detailu případu v detailu testovacího běhu	Stav testovacího případu byl změněn	Prošel
Změna stavu testovacího případu v detailu případu v detailu testovacího běhu	Stav testovacího případu byl změněn	Prošel
Změna stavu testovacího běhu detailu testovacího běhu	Stav testovacího běhu byl změněn	Prošel
Změna stavu testovacího běhu z přehledové tabulky testovacích běhů v detailu množiny	Stav testovacího běhu byl změněn	Prošel

Tabulka 7.5: Testovací sada Vytvoření a spuštění testovacích běhů.

Kapitola 8

Závěr

Cílem této práce bylo specifikovat a poté navrhnout a implementovat systém pro správu testů, který umožňuje řídit proces testování softwaru. Tento systém také tvoří zprostředkovatele pro nástroje platformy Testos a koncového uživatele. Na základě požadavků byl vytvořen model procesu testování. Tento model obsahuje správu testovaných projektů, správu testovacích požadavků, správu testovacích sad a testovacích případů, správu testovacích běhů a jejich spouštění a provádění.

Systém byl implementován jako webová aplikace a webová služba. Webová aplikace byla vytvořena pomocí aplikačního frameworku Laravel, který je postavený na programovacím jazyku PHP. Webová služba byla navržena podle architektonického stylu REST, poté byla dokumentována a implementována taktéž pomocí aplikačního frameworku Laravel.

Výsledný systém umožňuje řídit navržený proces testování a obsahuje základní správu uživatelů. Ověření funkcionality bylo dosaženo pomocí testování. Pro serverovou část webové aplikace a pro webovou službu byla vytvořena automatizovaná sadu testů, která pokrývá nutnou funkcionalitu systému. Klientská část webové aplikace byla testována ručně pomocí testovací sady, která pokryje základní funkčnost procesu testování.

8.1 Další rozvoj systému

Další vývoj systému se zaměří na:

- Pokročilejší správu týmů, uživatelů a rolí v systému tak, aby mohl administrátor testovaného projektu nastavit jednotlivým uživatelům role a práva k provádění některých operací.
- Integraci s nástroji třetích stran, zejména s nástroji pro evidenci chyb a problémů jako je například *JIRA*.
- Podporu automatizovaného spouštění testovacích běhů.
- Exportování informací o daném projektu ve formě testovacího plánu.

Literatura

[1] *15 Best Test Management Tools for Software Testers*. [Online; navštíveno 20.04.2017]. URL http://www.softwaretestinghelp.com/15-best-test-management-tools-for-software-testers/

[2] *Best Open Source Test Management Tools*. [Online; navštíveno 20.04.2017]. URL http://www.testingexcellence.com/best-open-source-test-management-tools/

[3] *Best Project Management Software and Tools*. [Online; navštíveno 15.04.2017]. URL http://www.capterra.com/project-management-software/

[4] *Manual testing – Wikipedia*. [Online; navštíveno 19.04.2017]. URL https://en.wikipedia.org/wiki/Manual_testing

[5] *Test Execution*. [Online; navštíveno 19.04.2017]. URL https://www.tutorialspoint.com/software_testing_dictionary/test_execution.htm

[6] *Webová aplikace*. [Online; navštíveno 20.04.2017]. URL https://cs.wikipedia.org/wiki/Webov%C3%A1_aplikace

[7] *Best Practices for Designing a Pragmatic RESTful API*. [Online; navštíveno 23.04.2017]. URL http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

[8] *Installation Laravel – The PHP Framework For Web Artisans*. [Online; navštíveno 22.04.2017]. URL https://laravel.com/docs/5.3

[9] *ISO/IEC/IEEE 29119-2: Test Processes*. [Online; navštíveno 09.05.2017]. URL http://ieeexplore.ieee.org/servlet/opac?punumber=6588541

[10] *Laravel – Wikipedia*. [Online; navštíveno 22.04.2017]. URL https://en.wikipedia.org/wiki/Laravel

[11] *PHP – Wikipedia*. [Online; navštíveno 20.04.2017]. URL https://cs.wikipedia.org/wiki/PHP

[12] *Python – Wikipedia*. [Online; navštíveno 22.04.2017]. URL https://cs.wikipedia.org/wiki/Python

[13] *Requirements management – Wikipedia*. [Online; navštíveno 16.04.2017].
URL https://en.wikipedia.org/wiki/Requirements_management

[14] *Test Management: The must-have functionality*. [Online; navštíveno 15.04.2017].
URL http://www.softwaretestingtools.com/test_management/test-management-must-have-functionality

[15] *Skupina Testos. Domovská stránka projektu Testos*. FIT VUT v Brně, 2017, [Online; navštíveno 11.05.2017].
URL http://testos.org

[16] *Skupina Testos. Nástroj pro tvorbu CEG grafů a tvorbu testovacích případů založených na CEG*. FIT VUT v Brně, 2017, [Online; navštíveno 11.05.2017].
URL https://pajda.fit.vutbr.cz/testos/ceg-editor

[17] Ammann, P.; Offutt, J.: *Introduction to Software Testing*. Cambridge University Press, 2008, ISBN 978-0-511-39330-3.

[18] Cohn, M.: *Succeeding with agile: software development using scrum*. Upper Saddle River: Addison-Wesley, 2010, ISBN 9780321579362.

[19] Haris, D.: *Nástroj pro hypertextovou dokumentaci testování*. Bakalářská práce, Vysoké učení technické v Brně. Fakulta informačních technologií, 2016, [Online; navštíveno 11.05.2017].
URL http://hdl.handle.net/11012/62248

[20] Havlát, M.: *TestLink – User Manual*. TestLink Communit, 2010, [Online; navštíveno 20.04.2017].
URL https://wiki.openoffice.org/w/images/1/1b/Testlink_user_manual.pdf

[21] Hruška, T.; Křivka, Z.: *Informační systémy – Studijní opora*. FIT VUT v Brně, Interní materiál, 2012, [Online; navštíveno 07.05.2017].
URL https://www.fit.vutbr.cz/study/courses/WAP/private/opory/OporaIIS2PISPojemDataProcesyTransakce.pdf

[22] Hynek, J.: *Informační dashboardy*. FIT VUT v Brně, Interní materiál, 2015, [Online; navštíveno 19.04.2017].
URL https://www.fit.vutbr.cz/study/courses/WAP/private/opory/PIS100Dashboard.pdf

[23] Jones, M.: *RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage*. 2012, [Online; navštíveno 30.04.2017].
URL https://tools.ietf.org/html/rfc6750

[24] Kimball, R.; Ross, M.: *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. 3th ed. John Wiley & Sons, Inc., 2013, ISBN 978-1-118-53080-1.

[25] Kosek, J.: *HTML – Tvorba dokonalých WWW stránek*. Grada Publishing, 1998, ISBN 80-7169-608-0.

[26] Křena, B.; Kočí, R.: *Úvod do softwarového inženýrství - Studijní opora*. FIT VUT v Brně, Interní materiál, Prosinec 2010, [Online; navštíveno 13.04.2017].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIUS-IT%2Ftexts%2FIUS_opora.pdf&cid=10027

[27] Patton, R.: *Testování softwaru*. Computer Press, 2001, ISBN 80-7226-636-5.

[28] Rychlý, M.: *Temporální a deduktivní databáze*. FIT VUT v Brně, 2015, [Online; navštíveno 26.04.2017].
URL http://www.fit.vutbr.cz/~rychly/public/docs/PDB.demo3-tmp_and_deductive/PDB.demo3-tmp_and_deductive.print.pdf

[29] Schildt, H.: *Java: The Complete Reference* . McGraw-Hill Education, 2014, ISBN 978-0-071-80855-2.

[30] Schmidt, D.; Gokhale, A.; Natarajan, B.: *Frameworks: Why They Are Important and How to Apply Them Effectively* . Department of Electrical Engineering and Computer Science Vanderbilt University Nashville, [Online; navštíveno 21.04.2017].
URL https://www.dre.vanderbilt.edu/~schmidt/PDF/Queue-04.pdf

[31] Smrčka, A.: *ITS – Úvod a pojmy v testování* Materiály k přednášce. FIT VUT v Brně, Interní materiál, 2016, [Online; navštíveno 13.04.2017].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FITS-IT%2Flectures%2F1-uvod.pdf&cid=10349

[32] Sommerville, I.: *Software engineering*. 9th ed. Boston: Pearson, 2011, ISBN 978-0-13-703515-1.

[33] van Veenendaal, E.: *Standard glossary of terms used in Software Testing* . International Software Testing Qualifications Board, Prosinec 2010, [Online; navštíveno 15.04.2017].
URL http://www.mstb.org/Downloadfile/ISTQB%20Glossary%20of%20Testing%20Terms%202.1.pdf

[34] Zendulka, J.; Rudolfová, I.: *Databázové systémy – Studijní opora*. FIT VUT v Brně, Interní materiál, 2006, [Online; navštíveno 23.04.2017].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIDS-IT%2Ftexts%2FIDS_predn.pdf&cid=10302

[35] Žára, O.: *Javascript – Programátorské techniky a webové technologie*. Computer press, 2015, ISBN 978-80-251-4573-9.

Přílohy

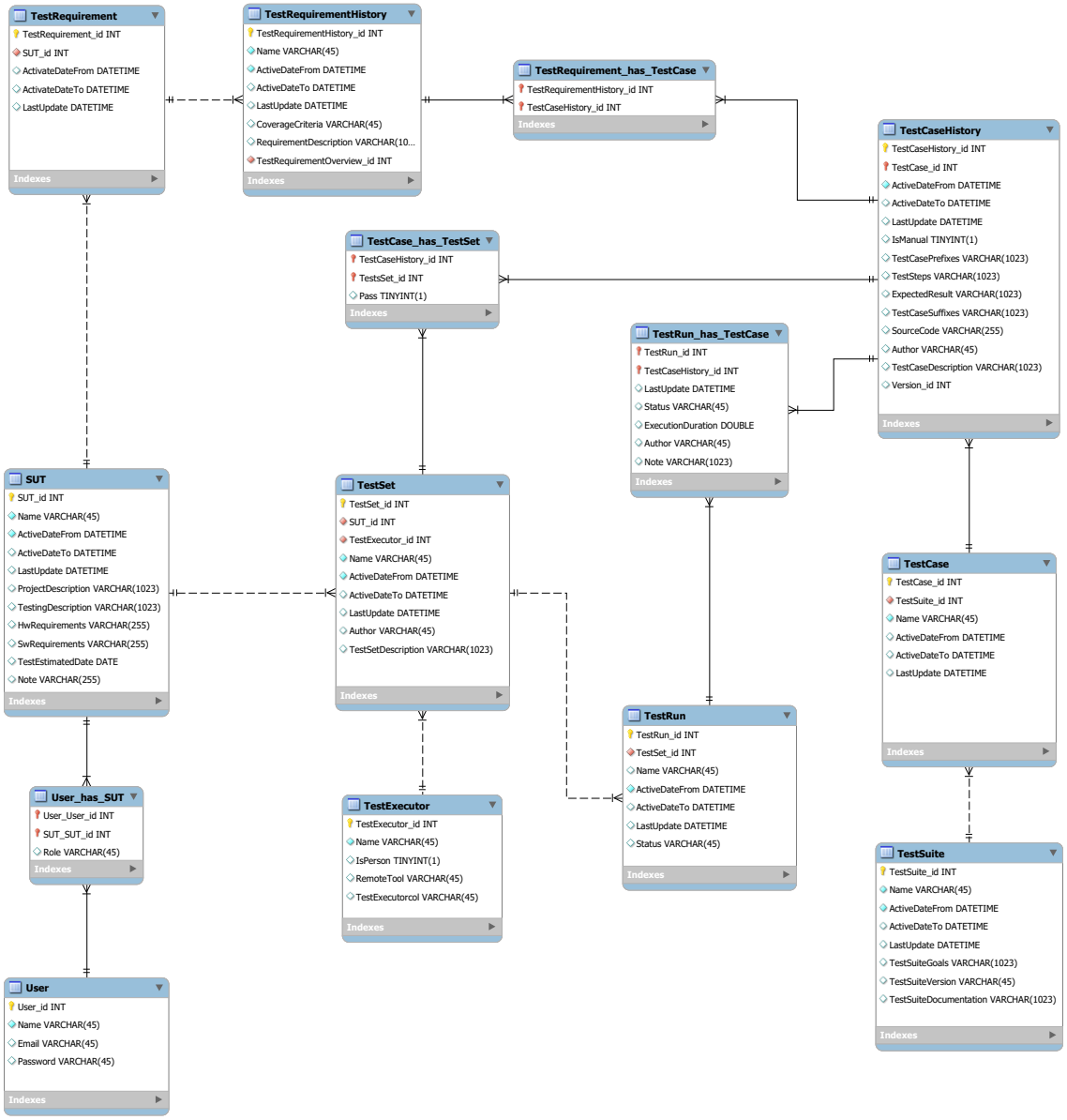
Příloha A

Obsah přiloženého paměťového média

- `/`
  - `src/` ..... adresář obsahující zdrojový kód informačního systému pro správu testů
  - `doc/`
    - `tex_src/` ..... adresář obsahující zdrojové soubory textu práce
    - `img_src/` . adresář obsahující zdrojové soubory pro obrázky používané v práci ve formátu xml. Tyto soubory lze otevřít a upravit ve webové aplikaci *draw.io*
    - `install.md` . soubor obsahující návod na manuální instalaci výsledného systému
    - `virtual.md` ..... návod pro spuštění aplikace skrze přiložený .ova soubor
    - `dsp.pdf` ..... dokument specifikace požadavků
  - `bp.pdf` ..... text bakalářské práce ve formátu pdf
  - `test-management-tool.ova` ..... soubor virtuálního stroje

Příloha B

Entitně-vztahový diagram



Obrázek B.1: Architektura aplikace.

Příloha C

Návod k manuální instalaci

Následující návod obsahuje postup, který pomůže uživateli ve zprovoznění výsledného systému na jeho lokálním PC.

Nutné prerekvizity

Návod počítá s tím, že má uživatel na svém lokálním PC nainstalován operační systém **unixového** typu. Na jiné platformě (např. Windows či macOS) lze systém taktéž zprovoznit. Je však možné, že formát jednotlivých příkazů se bude na této platformě lišit.

Dále je nutné mít na svém lokálním PC nainstalován:

- docker
- docker-compose
- wget nebo webový prohlížeč
- git
- unzip

Instalační postup

1. Získání zdrojových souborů a souborů nutných k provozu aplikace.

   ```
   wget http://www.stud.fit.vutbr.cz/~xnovak1k/tmt.zip
   ```

2. Rozbalení komprimovaného souboru a přístup do nově vzniklého adresáře.

   ```
   unzip tmt.zip
   cd tmt
   ```

3. Získání kopie zdrojových kódů projektu Laradock a přepnutí se na verzi, s níž návod počítá.

   ```
   git clone https://github.com/laradock/laradock.git
   git checkout f05512fd1d6676536c1920027f7a81675adcb461
   ```

4. Přístup do adresáře laradock a přejmenování souboru .env-example na .env.

   ```
   cd laradock
   cp env-example .env
   ```

5. Spustit kontejnery vyžadované aplikací. V případě prvního spuštění bude akce trvat delší dobu.

   ```
   docker-compose up -d nginx mysql
   ```

6. Přistup do kontejneru Workspace.

   ```
   docker-compose exec workspace bash
   ```

7. Aktualizace všech závislostí aplikace.

   ```
   composer update
   ```

8. Test, zdali vše proběhlo úspěšně. Většina testů by měla být úspěšná.

   ```
   phpunit
   ```

9. Vytvoření databáze.

   ```
   php artisan migrate
   ```

Pakliže všechny kroky proběhly úspěšně, nyní lze přistoupit k aplikaci například pomocí webového prohlížeče pod doménovým jménem localhost.

Příloha D

Návod ke spuštění výsledného systému skrze přiložený virtuální stroj

V přiloženém DVD se nachází soubor `test-management-tool.ova`, což je obraz virtuálního stroje, v němž je nainstalována výsledná aplikace bakalářské práce. Tento soubor lze spustit ve virtualizačním nástroji, například v nástroji *VirtualBox*.

Po spuštění virtuálního stroje je nutné se přihlásit jako uživatel *user*. Tento uživatel není chráněn heslem. Po úspěšném vstupu do operačního systému je nutné se dostat do adresáře *tmt/laradock*. V tomto adresáři poté spustit příkaz:

```
sudo docker-compose up -d nginx mysql
```

Pro autentizaci uživatele *root* se použije heslo *adminadmin*. Poté jde k aplikaci přistoupit přes webový prohlížeč *Firefox* po zadání doménového jména *localhost*. Pro vstup do aplikace se lze registrovat nebo využít připravený účet, který slouží k demonstračním účelům. Přístup k tomu demonstračnímu účtu je přes email *admin@admin.com* a heslem *123456*.

Příloha E

Dokumentace k REST API

Přehled

Následující příloha poskytuje dokumentaci k aplikačnímu rozhraní k informačnímu systému pro správu testů. Rozhraní je postavené na principu RESTu, mluvíme tedy o REST API.

V následujících částech bude dokumentován serializační formát, autentizace k API a dále jednotlivé zdroje (angl. resources), s nimiž jde pomocí API zacházet.

Serializační formát

Aplikace podporuje příjem požadavků ve formátu JSON. Aplikace odpovídá rovněž ve formátu JSON. Jiné formáty nebudou akceptovány.

Adresace

Standardní adresace API je *host/api/v**X**/resource*, kde x je číslo verze daného API.

Autentizace

Autentizace je implementovaná pomocí tzn. API tokenu, který je předáván při každém požadavku do aplikace. Token je předáván v **HTTP** hlavičce v poli *Authorization* pomocí *Bearer Token*. Viz RFC 6750 [23].

API token je generován každému uživateli při registraci a skládá se z 60 alfanumerických znaků. Uživatel ho poté nalezne na stránce *User settings*.

Příklad HTTP hlavičky

```
GET /api/v1/testcases/1 HTTP/1.1
Host: localhost:8000
Authorization: Bearer faQIfZxazJK5tYx40bbsEAG7G2Ab7zzlkLQ1PDp9BRF2I3RHRDkM7VSYP3Rj
Cache-Control: no-cache
```

V případě požadavku typu POST a PUT je potřeba zahrnout i další pole:

```
Content-Type: application/json
```

Příklad cURL volání

```
curl --request GET \
--url http://localhost:8000/api/v1/testcases/1 \
--header 'authorization: Bearer faQIfZxazJK5tYx40bbsEAG7G2A \
b7zzlkLQ1PDp9BRF2I3RHRDkM7VSYP3Rj' \
--header 'cache-control: no-cache' \
```

V případě nepovedené autentizace je uživateli odeslána odpověď s návratovým HTTP kódem **401** a obsahem:

```
{
"error" : "Unauthenticated."
}
```

Testovací sada

GET /api/v1/testsuites

Popis

Vrátí všechny testovací sady

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"TestSuite_id": "number",
"Name": "string",
"TestSuiteGoals": "string",
"TestSuiteVersion": "string",
"TestSuiteDocumentation": "string",
"href": "string"
}
```

GET /api/v1/testsuites/{id}

Popis

Vrátí vybranou testovací sadu

Parametry

id - identifikátor testovací sady

Odpovědi

1. Úspěch - návratový kód: 200
Obsah:

```
{
"TestSuite_id": "number",
"Name": "string",
"TestSuiteGoals": "string",
"TestSuiteVersion": "string",
"TestSuiteDocumentation": "string",
"href": "string"
}
```

POST /api/v1/testsuites

Popis

Uloží testovací sadu do systému

Vstup

```
{
"*Name": "string",
"TestSuiteGoals": "string",
"TestSuiteVersion": "string",
"TestSuiteDocumentation": "string"
}
```

Poznámka: Atributy označené * jsou povinné

Odpovědi

1. Úspěch - návratový kód: 201
Obsah:

```
{
"TestSuite_id": "number",
"Name": "string",
"TestSuiteGoals": "string",
"TestSuiteVersion": "string",
"TestSuiteDocumentation": "string",
"href": "string"
}
```

2. Neúspěch - návratový kód: 400
Popis: V případě nezadaného jména, nebo v případě, kdy jméno překročí maximální hranici 45 znaků
Obsah:

```
{
"error": "Test suite name error"
}
```

PUT /api/v1/testsuites/{id}

Popis

Edituje existující testovací sadu

Parametry

id - identifikátor testovací sady

Vstup

```
{
"Name": "string",
"TestSuiteGoals": "string",
"TestSuiteVersion": "string",
"TestSuiteDocumentation": "string"
}
```

Odpovědi

1. Úspěch - návratový kód: 201
Obsah:

```
{
"TestSuite_id": "number",
"Name": "string",
"TestSuiteGoals": "string",
"TestSuiteVersion": "string",
"TestSuiteDocumentation": "string",
"href": "string"
}
```

2. Neúspěch - návratový kód: 404
Popis: V případě nenalezení testovací sady
Obsah:

```
{
"error": "Testsuite not found"
}
```

DELETE /api/v1/testsuites/{id}

Popis

Archivuje danou testovací sadu

Parametry

id - identifikátor testovací sady

Odpovědi

1. Úspěch - návratový kód: 200
Obsah:

```
{
"success": "Deleted"
}
```

2. Neúspěch - návratový kód: 404
Popis: V případě nenalezení testovací sady
Obsah:

```
{
"error": "Testsuite not found"
}
```

Testovací případ

GET /api/v1/testcases

Popis

Vrátí všechny testovací případy

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:


```
{
"TestCase_id": "number",
"TestSuite_id": "number",
"Name": "string",
"IsManual": 0,
"TestCasePrefixes": "string",
"TestSteps": "string",
"ExpectedResult": "string",
"TestCaseSuffixes": "string",
"SourceCode": "string",
"TestCaseDescription": "string",
"Note": "string",
"href": "string"
}
```


GET /api/v1/testcases/{id}

Popis

Vrátí vybraný testovací případ

Parametry

id - identifikátor testovacího případu

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:


```
{
"TestCase_id": "number",
"TestSuite_id": "number",
"Name": "string",
"IsManual": 0,
"TestCasePrefixes": "string",
"TestSteps": "string",
"ExpectedResult": "string",
"TestCaseSuffixes": "string",
"SourceCode": "string",
"TestCaseDescription": "string",
"Note": "string",
"href": "string"
}
```


GET /api/v1/testsuites/{idSuite} /testcases

Popis

Vrátí všechny testovací případy pro danou testovací sadu

Parametry

idSuite - identifikátor testovací sady

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:


```
{
"TestCase_id": "number",
"TestSuite_id": "number",
"Name": "string",
"IsManual": 0,
"TestCasePrefixes": "string",
"TestSteps": "string",
"ExpectedResult": "string",
"TestCaseSuffixes": "string",
"SourceCode": "string",
"TestCaseDescription": "string",
"Note": "string",
"href": "string"
}
```


2. Neúspěch: návratový kód: 404
Obsah:


```
{
"error": "Testsuit not found."
}
```


POST /api/v1/testcases

Popis

Uloží nový testovací případ do systému

Vstup

```
{
"TestSuite_id": "number",
"*Name": "string",
"IsManual": 0,
"TestCasePrefixes": "string",
"TestSteps": "string",
"ExpectedResult": "string",
"TestCaseSuffixes": "string",
"TestCaseDescription": "string",
"Note": "string"
}
```

Poznámka: Atributy označené * jsou povinné

Odpovědi

1. Úspěch: návratový kód: 201
Obsah:

```
{
"TestCase_id": "number",
"TestSuite_id": "number",
"Name": "string",
"IsManual": 0,
"TestCasePrefixes": "string",
"TestSteps": "string",
"ExpectedResult": "string",
"TestCaseSuffixes": "string",
"SourceCode": "string",
"TestCaseDescription": "string",
"Note": "string",
"href": "string"
}
```

2. Neúspěch: návratový kód: 404
Popis: V případě zadání neexistující TestSuiteId
Obsah:

```
{
"error": "Test suite doesn't exists"
}
```

3. Neúspěch: návratový kód: 400
Popis: V případě nezadání atributu Name či překročení maximální povolené velikosti 45 znaků
Obsah:

```
{
"error": "Test case name error"
}
```

PUT /api/v1/testcases/{id}

Popis

Edituje existující testovací případ

Parametry

id - identifikátor testovacího případu

Vstup

```
{
"TestSuite_id": "number",
"Name": "string",
"IsManual": 0,
"TestCasePrefixes": "string",
"TestSteps": "string",
"ExpectedResult": "string",
"TestCaseSuffixes": "string",
"TestCaseDescription": "string",
"Note": "string"
}
```

Odpovědi

1. Úspěch: návratový kód: 201
Obsah:

```
{
"TestSuite_id": "number",
"Name": "string",
"TestSuiteGoals": "string",
"TestSuiteVersion": "string",
"TestSuiteDocumentation": "string",
"href": "string"
}
```

2. Neúspěch: návratový kód: 400
Obsah:

```
{
"error": "TestCase not found"
}
```

DELETE /api/v1/testcases/{id}

Popis

Archvivuje daný testovací případ

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"success": "Deleted"
}
```

2. Neúspěch: návratový kód: 404
Popis: V případě nenalezení testovacího případu
Obsah:

```
{
"error": "TestCase not found"
}
```

Projekt

GET /api/v1/projects

Popis

Vrátí všechny projekty přiřazené danému uživateli

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"SUT_id": "number",
"Name": "string",
"ActiveDateFrom": "string",
"ActiveDateTo": "string",
"LastUpdate": "string",
"ProjectDescription": "string",
"TestingDescription": "string",
"HwRequirements": "string",
"SwRequirements": "string",
"TestEstimatedDate": "string",
"Note": "string",
"Role" : "string"
}
```

GET /api/v1/projects/{id}

Popis

Vrátí daný projekt uživateli

Parametry

id - identifikátor projektu

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"SUT_id": "number",
"Name": "string",
"ActiveDateFrom": "string",
"ActiveDateTo": "string",
"LastUpdate": "string",
"ProjectDescription": "string",
"TestingDescription": "string",
"HwRequirements": "string",
"SwRequirements": "string",
"TestEstimatedDate": "string",
"Note": "string",
"Role" : "string"
}
```

2. Neúspěch - návratový kód: 404
Popis: projekt nebyl nalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch - návratový kód: 400
Popis: uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

POST /api/v1/projects

Popis

Uloží projekt do aplikace

Vstup

```
{
"*Name": "string",
"ActiveDateFrom": "string",
"ActiveDateTo": "string",
"LastUpdate": "string",
"ProjectDescription": "string",
"TestingDescription": "string",
"HwRequirements": "string",
"SwRequirements": "string",
"TestEstimatedDate": "string",
"Note": "string",
}
```

Poznámka: Atributy označené * jsou povinné

Odpovědi

1. Úspěch: návratový kód: 201
Obsah:

```
{
"SUT_id": "number",
"Name": "string",
"ActiveDateFrom": "string",
"ActiveDateTo": "string",
"LastUpdate": "string",
"ProjectDescription": "string",
"TestingDescription": "string",
"HwRequirements": "string",
"SwRequirements": "string",
"TestEstimatedDate": "string",
"Note": "string",
}
```

2. Neúspěch - návratový kód: 400
Popis: V případě nezadání atributu Name či překročení maximální povolené velikosti 45 znaků
Obsah:

```
{
"error": "SUT name error"
}
```

PUT /api/v1/projects/{id}

Popis

Edituje existující projekt

Parametry

id - identifikátor projektu

Vstup

```
{
"Name": "string",
"ActiveDateFrom": "string",
"ActiveDateTo": "string",
"LastUpdate": "string",
"ProjectDescription": "string",
"TestingDescription": "string",
"HwRequirements": "string",
"SwRequirements": "string",
"TestEstimatedDate": "string",
"Note": "string",
}
```

Odpovědi

1. Úspěch: návratový kód: 201
Obsah:

```
{
"SUT_id": "number",
"Name": "string",
"ActiveDateFrom": "string",
"ActiveDateTo": "string",
"LastUpdate": "string",
"ProjectDescription": "string",
"TestingDescription": "string",
"HwRequirements": "string",
"SwRequirements": "string",
"TestEstimatedDate": "string",
"Note": "string",
}
```

2. Neúspěch - návratový kód: 404
Popis: V případě nezadání atributu Name či překročení maximální povolené velikosti 45 znaků
Obsah:

```
{
"error": "SUT not found"
}
```

DELETE /api/v1/projects/{id}

Popis

Archivuje daný projekt

Parametry

id - identifikátor projektu

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"success": "Deleted"
}
```

2. Neúspěch - návratový kód: 404
Popis: V případě nenalezení testovací sady
Obsah:

```
{
"error": "Project not found"
}
```

Testovací požadavky

GET /api/v1/projects/{projectId} /requirements

Popis

Zobrazí všechny požadavky k danému projektu včetně testovacích případů, které daný požadavek pokrývají.

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"TestRequirement_id": "number",
"SUT_id": "number",
"Name": "string",
"CoverageCriteria": "string",
"RequirementDescription": "string",
"TestCase": [
    {
       "TestCase_id": "number",
       "Name": "string"
    }
]
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

GET /api/v1/projects/{projectId} /requirements/{reqId}

Popis

Zobrazí detail požadavku k danému projektu včetně testovacích případů, které daný požadavek pokrývají.

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"TestRequirement_id": "number",
"SUT_id": "number",
"Name": "string",
"CoverageCriteria": "string",
"RequirementDescription": "string",
"TestCase": [
    {
       "TestCase_id": "number",
       "Name": "string"
    }
]
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

4. Neúspěch: návratový kód: 400
Popis: Neexistující požadavek
Obsah:

```
{
"error": "No requirement found"
}
```

POST /api/v1/projects/{projectId} /requirements

Popis

Uloží požadavek

Vstup

```
{
"*Name": "string",
"CoverageCriteria": "string",
"RequirementDescription": "string"
}
```

Poznámka: Atributy označené * jsou povinné

Odpovědi

1. Úspěch: návratový kód: 201
Obsah:

```
{
"TestRequirement_id": "number",
"Name": "string",
"SUT_id": "number",
"CoverageCriteria": "string",
"RequirementDescription": "string",
"LastUpdate": "string",
"ActiveDateFrom": "string"
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

PUT /api/v1/projects/{projectId} /requirements/{reqId}

Popis

Upraví požadavek.

Vstup

```
{
"Name": "string",
"CoverageCriteria": "string",
"RequirementDescription": "string"
}
```

Odpovědi

1. Úspěch: návratový kód: 201
Obsah:

```
{
"TestRequirement_id": "number",
"Name": "string",
"SUT_id": "number",
"CoverageCriteria": "string",
"RequirementDescription": "string",
"LastUpdate": "string",
"ActiveDateFrom": "string"
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

4. Neúspěch: návratový kód: 400
Popis: Neexistující požadavek
Obsah:

```
{
"error": "No requirement found"
}
```

DELETE /api/v1/projects/{projectId} /requirements/{reqId}

Popis

Archivuje požadavek.

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"success": "Deleted"
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

4. Neúspěch: návratový kód: 400
Popis: Neexistující požadavek
Obsah:

```
{
"error": "No requirement found"
}
```

Množina testovacích běhů

GET /api/v1/projects/{projectId} /testsets

Popis

Zobrazí všechny aktivní množiny testovacích běhů. Včetně testovacích případů, které jsou v dané množině zahrnuty.

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"TestSet_id": "number",
"SUT_id": "number",
"Name": "string",
"Author": "string",
"TestSetDescription": "string",
"TestCase": [
    {
       "TestCase_id": "number",
       "Name": "string"
    }
]
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

GET /api/v1/projects/{projectId} /testsets/{setId}

Popis

Zobrazí vybranou množinu včetně testovacích případů, které jsou v dané množině zahrnuty.

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"TestSet_id": "number",
"SUT_id": "number",
"Name": "string",
"Author": "string",
"TestSetDescription": "string",
"TestCase": [
    {
       "TestCase_id": "number",
       "Name": "string"
    }
]
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

4. Neúspěch: návratový kód: 404
Popis: Množina testovacích běhů nebyla nalezena
Obsah:

```
{
"error": "Not existing set"
}
```

POST /api/v1/projects/{projectId} /testsets

Popis

Vytvoří množinu testovacích běhů a naplní ho testovacími případy.

Vstup

```
{
"*Name": "string",
"Author": "string",
"TestSetDescription": "string",
"*TestCases*": ["number"]
}
```

Poznámka: Atributy označené * jsou povinné

Příklad

```
"Name": "Validace",
"Author": "Admin",
"TestSetDescription": "popis",
"TestCases": [1, 2, 5, 10]
```

Odpovědi

1. Úspěch: návratový kód: 201
Obsah:

```
{
"TestSet_id": "number",
"SUT_id": "number",
"Name": "string",
"Author": "string",
"TestSetDescription": "string",
"TestCase": [
    {
       "TestCase_id": "number",
       "Name": "string"
    }
]
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

4. Neúspěch: návratový kód: 400
Popis: V případě nezadání atributu Name či překročení maximální povolené velikosti 45 znaků
Obsah:

```
{
"error": "Set name error"
}
```

PUT /api/v1/projects/{projectId} /testsets/{setId}

Popis

Aktualizuje množinu testovacích běhů

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"TestSet_id": "number",
"SUT_id": "number",
"Name": "string",
"Author": "string",
"TestSetDescription": "string",
"TestCase": [
    {
       "TestCase_id": "number",
       "Name": "string"
    }
]
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

4. Neúspěch: návratový kód: 404
Popis: Množina testovacích běhů nebyla nalezena
Obsah:

```
{
"error": "Not existing set"
}
```

DELETE /api/v1/projects/{projectId}/testsets/{setId}

Popis

Archivuje množinu testovacích běhů

Odpovědi

1. Úspěch: návratový kód: 201
Obsah:

```
{
"success": "Deleted"
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

4. Neúspěch: návratový kód: 404
Popis: Množina testovacích běhů nebyla nalezena
Obsah:

```
{
"error": "Not existing set"
}
```

Testovací běhy

GET /api/v1/projects/{projectId} /testsets/{setId}/testruns

Popis

Zobrazí všechny testovací běhy náležící dané množině

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"TestRun_id": "number",
"TestSet_id": "number",
"Status": "string",
"TestCase": [
   {
      "TestCase_id": "number",
      "Name": "string",
      "Status": "string"
   }
]
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

4. Neúspěch: návratový kód: 404
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

3. Neúspěch: návratový kód: 400
Popis: Množina testovacích běhů nebyla nalezena
Obsah:

```
{
"error": "Testset don't exist"
}
```

GET /api/v1/projects/{projectId} /testsets/{setId}/testruns/{runId}

Popis

Zobrazí daný testovací běh.

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"TestRun_id": "number",
"TestSet_id": "number",
"Status": "string",
"TestCase": [
   {
      "TestCase_id": "number",
      "Name": "string",
      "Status": "string"
   }
]
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

4. Neúspěch: návratový kód: 404
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

3. Neúspěch: návratový kód: 400
Popis: Množina testovacích běhů nebyla nalezena
Obsah:

```
{
"error": "Testset don't exist"
}
```

POST /api/v1/projects/{projectId} /testsets/{setId}/testruns

Popis

Vytvoří nový testovací běh

Vstup

Žádný vstup není zpracován. Pouhé zaslaní POST se správnými parametry v URL vytvoří nový testovací běh

Odpovědi

1. Úspěch: návratový kód: 201
Obsah:

```
{
"TestRun_id": "number",
"TestSet_id": "number",
"Status": "string",
"TestCase": [
   {
      "TestCase_id": "number",
      "Name": "string",
      "Status": "string"
   }
]
}
```

Poznámka: pro status jsou povolené hodnoty: "Running", "Finished", "Archived"
2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

4. Neúspěch: návratový kód: 404
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "Not rights to project"
}
```

3. Neúspěch: návratový kód: 400
Popis: Množina testovacích běhů nebyla nalezena
Obsah:

```
{
"error": "Testset don't exist"
}
```

PUT /api/v1/projects/{projectId} /testsets/{setId}/testruns/{runId}

Popis

Aktualizuje stav daneho testovacího běhu

Vstup

```
{
"Status" : "string"
}
```

Poznámka: pro status jsou povolené hodnoty: "Running", "Finished", "Archived"

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"TestRun_id": "number",
"TestSet_id": "number",
"Status": "string",
"TestCase": [
   {
      "TestCase_id": "number",
      "Name": "string",
      "Status": "string"
   }
]
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

4. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

3. Neúspěch: návratový kód: 404
Popis: Množina testovacích běhů nebyla nalezena
Obsah:

```
{
"error": "Testset don't exist"
}
```

DELETE /api/v1/projects /{projectId}/testsets/{setId} /testruns/{runId}

Popis

Archivuje dany testovací běh

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"success": "Deleted"
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

4. Neúspěch: návratový kód: 404
Popis: Množina testovacích běhů nebyla nalezena
Obsah:

```
{
"error": "Testset don't exist"
}
```

5. Neúspěch: návratový kód: 404
Popis: Testovací běh nebyl nalezen
Obsah:

```
{
"error": "Test run don't exist"
}
```

PUT /api/v1/projects/{projectId} /testsets/{setId}/testruns/{runId} /testcase/{testCaseID}

Popis

Změní status testovaciho případu v rámci testovacího běhu

Vstup

```
{
"Status" : "string"
}
```

Poznámka: pro status jsou povolené hodnoty: "Pass", "Fail", "Blocked"

Odpovědi

1. Úspěch: návratový kód: 200
Obsah:

```
{
"TestRun_id": "number",
"TestSet_id": "number",
"Status": "string",
"TestCase": [
   {
      "TestCase_id": "number",
      "Name": "string",
      "Status": "string"
   }
]
}
```

2. Neúspěch: návratový kód: 404
Popis: Projekt nenalezen
Obsah:

```
{
"error": "Project not found"
}
```

3. Neúspěch: návratový kód: 400
Popis: Uživatel nemá práva k projektu
Obsah:

```
{
"error": "No rights to project"
}
```

4. Neúspěch: návratový kód: 404
Popis: Množina testovacích běhů nebyla nalezena
Obsah:

```
{
"error": "Testset don't exist"
}
```

5. Neúspěch: návratový kód: 404
Popis: Testovací běh nebyl nalezen
Obsah:

```
{
"error": "Test run don't exist"
}
```