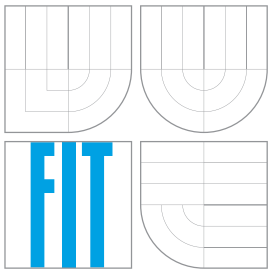
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRE HYPERTEXTOVÚ DOKUMENTÁCIU TESTOVANIA

GENERATION OF HYPERTEXT DOCUMENTATION OF TESTING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DANIEL HARIS

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ALEŠ SMRČKA, Ph.D.

BRNO 2016

Zadání bakalářské práce/17199/2015/xharis00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Haris Daniel**
Obor: Informační technologie
Téma: **Nástroj pro hypertextovou dokumentaci testování**
Generation of Hypertext Documentation of Testing
Kategorie: Softwarové inženýrství

Pokyny:

1. Nastudujte jazyky pro vytváření hypertextů (HTML, XML, PDF). Nastudujte testování založené na požadavcích (Requirement-based testing).
2. Navrhněte nástroj pro vytváření hypertextových reportů pro testovací účely. Reporty budou obsahovat uživatelem specifikované popisy jednotlivých testovacích případů, odkazy na požadavky spojené s testovacími případy a odkazy do zdrojových kódů testovacích případů. Nástroj bude vytvářet přehledový report o aktuálním pokrytí specifikace požadavků dokumentovanými testy.
3. Implementujte navržený nástroj s webovým rozhraním. Nástroj bude umožňovat export reportu do HTML.
4. Demonstrujte funkčnost nástroje na netriviálním případu užití.

Literatura:

- Whalen, M. W., Rajan, A., Heimdahl, M. P. E., Miller, S. P.: Coverage Metrics for Requirementrs-based Testing, ISSTA, s. 25-36, 2006. DOI: 10.1145/1146238.1146242
- Studijní materiály ITS, přednáška Testování na základě specifikace požadavků, 2014.

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 2015
Datum odevzdání: 18. května 2016

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

Cieľom tejto bakalárskej práce je návrh a implementácia pomocného nástroja pre testovanie na základe špecifikácie požiadaviek. Nástroj sa skladá z webovej aplikácie, ktorá sa zameriava na vizualizáciu pokrytia špecifikácie požiadaviek jednotlivými testovacími prípadmi. Ďalšou časťou nástroja je agent, ktorý slúži na generovanie testovacích šablón na základe informácií zo samotného nástroja, na spúšťanie testovacích sád a na poskytnutie výsledkov testovania. Výstupom tohto nástroja je prehľadový report o aktuálnom pokrytí špecifikácie požiadaviek.

Abstract

The objective of this bachalor's thesis is to design and implement a helper tool for requirements-based testing. Tool consists of a web application, which focuses on visualization of coverage of requirements specification by user defined test cases. Another part of this tool is an agent, which generates templates of test suites based on information from the tool itself, executes test suites and returns test execution results. Output of this tool is a summary report about actual coverage of requirements specification.

Kľúčové slová

testovanie na základe požiadaviek, špecifikácia požiadaviek, JavaScript, Meteor.js, Node.js, Python unittest, REST API

Keywords

requirements-based testing, requirements specification, JavaScript, Meteor.js, Node.js, Python unittest, REST API

Citácia

HARIS, Daniel. *Nástroj pre hypertextovú dokumentáciu testovania.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce SMRČKA ALEŠ.

Nástroj pre hypertextovú dokumentáciu testovania

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Aleša Smrčky, PhD. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
DANIEL HARIS
18. mája 2016

Poďakovanie

Ďakujem svojmu vedúcemu práce Ing. Alešovi Smrčkovi, Ph.D. za odborné vedenie tejto práce, za jeho cenné rady, ochotu pri konzultáciách a za konštruktívne pripomienky.

Obsah

Kapitola 1

Úvod

V poslednej dobe sa kladie veľký dôraz na skoré odhalenie chýb pri vývoji softvéru (či už pri vodopádovom modeli alebo agilných metodikách). Štúdie ukázali, že hlavnou príčinou softvérových chýb sú špecifikácie požiadaviek (angl. *requirements specification*). Podľa [8] viac než 50 % softvérových chýb má prvotnú príčinu v neúplnej, nesprávnej, nejednoznačnej alebo logicky nekonzistentnej špecifikácie požiadaviek. Jej tvorba sa často necháva až na neskoršie fázy vývoja, môžu sa veľmi často meniť alebo ju dostatočne neprebrali všetci členovia tímu. Problémom je, že sa postupom času zvyšujú náklady na odhalenie chýb [10], preto je dôležité venovať dfiostatok času a úsilia na vznik kvalitnej špecifikácie požiadaviek.

Téma práce

V tejto práci sa zameriame na testovanie na základe požiadaviek (angl. *requirements-based testing*), ktoré je tvorené 2 fázami – procesom skvalitnenia špecifikácií a tvorbou testovacích prípadov na ich pokrytie [9]. Význam testovania na základe požiadaviek spočíva v odstránení chýb už v prvotných fázach vývoja softvéru, čím sa viacnásobne znížia náklady na testovanie v neskorších fázach vývoja softvéru.

Cieľom tejto bakalárskej práce je návrh a implementácia pomocného nástroja pre testovanie na základe požiadaviek. Nástroj sa zameriava na vizualizáciu pokrytia špecifikácie vytvorenými testovacími prípadmi. Súčasťou nástroja je aj agent, ktorý umožňuje generovanie testovacích šablón podľa vytvorených testovacích prípadov z nástroja. Ďalej slúži na spúšťanie testovacích sád a na získanie informácií o výsledkoch testovania. Výstupom tohto nástroja je report o aktuálnom pokrytí špecifikácie.

Motivácia

Hlavnou motiváciou na vznik tejto práce je nedostatok existujúcich nástrojov tejto domény, či ide o podporné nástroje na skvalitnenie špecifikácií alebo nástrojov na generovanie testovacích prípadov na ich pokrytie. Medzi existujúce nástroje patrí komerčný nástroj *BenderRBT*[1], ktorý sa zameriava na tvorbu minimálnych testovacích sád na pokrytie špecifikácií. Ďalším komerčným softvérom na správu špecifikácií a testovacích prípadov je *Jama Contour*[2].

[1] BenderRBT – http://www.benderrbt.com/bendersoftware.htm

[2] Contour of Jama – http://www.jamasoftware.com/jama-contour-is-jama/

Štruktúra dokumentu

Kapitola 2 obsahuje teoretický rozbor testovania na základe požiadaviek. V kapitole 3 sa nachádza špecifikácia požiadaviek nástroja a jeho kompletný návrh. Návrh je popísaný z 3 pohľadov – návrh architektúry, návrh funkcionality a návrh grafického užívateľského rozhrania. Implementácia nástroja je detailne popísaná v kapitole 4, na začiatku ktorej sa nachádza popis použitých technológií. Demonštrácia funkčnosti implementovaného nástroja sa nachádza v kapitole 5.

Kapitola 2

Testovanie na základe požiadaviek

> A problem well stated is a problem half-solved.
>
> Charles Kettering

Táto kapitola sa zaoberá teoretickým rozborom zadaním práce. Sekcia 2.1 popisuje, čo je testovanie softvéru, aké základné prístupy a spôsoby testovania poznáme. Na konci tejto sekcie popíšeme, do ktorej oblasti testovania patrí testovanie na základe požiadaviek. V sekcii 2.2 si definujeme pojem špecifikácie požiadaviek. Ďalej v sekcii 2.3 popíšeme význam testovania na základe požiadaviek a na konci kapitoly v sekcii 2.4 predstavíme dôležité techniky tohto testovania používané v praxi .

2.1 Testovanie softvéru

Testovanie softvéru je súbor procesov slúžiacich na kontrolu kvality softvéru, ktorých cieľom je dosiahnutie požadovanej kvality softvéru z hľadiska funkčnosti, spoľahlivosti, výkonnosti, použiteľnosti a podporovateľnosti. Kontrola kvality sa môže uskutočniť či už pre jednotlivé časti systému alebo pre systém ako celok. Existujú 2 prístupy pri testovaní softvéru:

- **Systémové testovanie** (angl. *system testing*) – tester má informácie o tom, čo má predložený softvér robiť, resp. aké má vstupy a očakávané výstupy. Nemá informácie o jeho vnútornom fungovaní a štruktúre. Tester dokáže predložený systém iba pozorovať.
- **Štrukturálne testovanie** (angl. *structural testing*) – komplementárne voči systémovému testovaniu. Je založené na analýze vnútornej štruktúry a fungovaní softvéru.

Pri testovaní softvéru existujú 2 spôsoby testovania:

- **Statické testovanie** – softvér je testovaný bez spustenia jeho zdrojových kódov. Nad skúmaným objektom vykonávame analýzu, revíziu alebo kontrolu.
- **Dynamické testovanie** – testovaný objekt spustíme a analyzujeme počas jeho behu.

V tejto práci sa zameriame na statické systémové testovanie, ktoré sa nazýva testovanie na základe požiadaviek [10]. Zaoberá sa testovaním špecifikácií požiadaviek, teda dokumentom obsahujúcim funkcionálne požiadavky určitého softvéru. Je tvorené 2 fázami – procesom skvalitnenia špecifikácií a tvorbou testovacích prípadov na ich pokrytie. Štandard,

v ktorom je obsiahnuté aj toto testovanie má názov *ISO/IEC/IEEE 29119-1*[1]. Konkrétne sa testovaniu špecifikácií požiadaviek venuje kapitola *5.6.2* v štandarde.

2.2 Špecifikácia požiadaviek

Pod pojmom špecifikácia požiadaviek rozumieme dokument alebo sadu dokumentov, ktorá obsahuje funkcionálne požiadavky na predložený softvér. Ideálne je, ak sú požiadavky nejakým spôsobom formalizované alebo aspoň štrukturované, kde každá časť štruktúry má svoj identifikátor a je rozdelená na menšie celky, ktoré sú označené. Funkcionálne požiadavky popisuje pomocou slov, obrázkov, grafov, diagramov a pod. Je dôležité poznamenať, že špecifikácia by mala popisovať, **čo** daný softvér robí a nie akým spôsobom to robí. Štandard *IEEE 830-1998*[2] popisuje požadovaný obsah špecifikačného dokumentu a všetky jeho aspekty.

Požiadavky dôrazne premysleného špecifikačného dokumentu, v ktorej je všetko tak, ako má byť, musia mať nasledujúce vlastnosti [10]:

- **deterministické** – vstupy systému jasne definujú jeho výstupy,
- **úplné** – zahrňuje všetky požiadavky a obsahuje všetky prípady použitia a vzťahy medzi tvrdeniami,
- **správne** – navrhnuté riešenie je správne bez chýb a definuje správne požadovaný cieľ,
- **jednoznačné** – požiadavky majú iba jednu interpretáciu,
- **konzistentné** – požiadavky sú napísané jednotným štýlom,
- **neredundantné** – nepopisujú tú istú sadu funkcií alebo udalosti viackrát, ale používajú identifikátory a referencie,
- **explicitné** – informácie, ktoré nie sú obsiahnuté v špecifikácii sa nesmú automaticky predpokladať,
- **stručné** – viac informácií implikuje aj viac chýb.

Tvorba špecifikácií vyžaduje veľké množstvo energie a času, keďže je to náročný proces. Oneskoruje začatie implementácie, kvôli čomu sa na tvorbu špecifikácií častokrát nekladie veľký dôraz a programátorom sa môže zdať proces tvorby špecifikácií „zdržujúci“ pri vývoji softvéru. Jedinou možnosťou zaistenia, že konečný produkt vyhovuje požiadavkám zákazníka, je dôkladný popis celého produktu v podobe špecifikácie. Je to taktiež jedinou možnosťou naplánovania potrebného testovania. Testeri následne majú k dispozícii podrobný popis výsledného produktu a dokážu odhaliť prvé chyby ešte pred samotnou implementáciou. Všeobecne platí, že čas strávený tvorbou špecifikácie, ušetrí niekoľkonásobne viac času stráveným hľadaním chýb v implementačnej alebo postimplementačnej fáze.

Existuje niekoľko dôvodov podľa [4], prečo je tvorba kvalitnej špecifikácie rozumným nápadom:

- dobrý spôsob porozumenia požiadaviek na výsledný produkt,

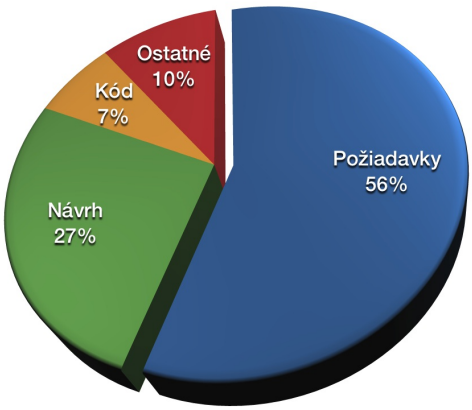
[1] ISO/IEC/IEEE 29119-1 – http://www.softwaretestingstandard.org/part1.php
[2] IEEE 830-1998 – https://standards.ieee.org/findstds/standard/830-1998.html

- pomáha zákazníkovi pri validácii výsledného produktu, že naozaj spĺňa jeho požiadavky,
- definuje a objasňuje, čo sa musí implementovať, aby predložený softvér spĺňal stanovené požiadavky,
- umožňuje deriváciu testovacích prípadov a očakávaných výsledkov, ktoré umožňujú verifikáciu, že implementácia daného softvéru vyhovuje stanoveným špecifikáciám.

Napriek výhodám tvorby špecifikácie pred samotnou implementáciou, nie v každom projekte sa zohľadní jej tvorba a to z rozličných technických alebo iných dôvodov.

2.3 Význam testovania na základe požiadaviek

Jeho význam spočíva v odstránení všetkých možných chýb už v prvotných fázach vývoja softvéru, čím sa viacnásobne znížia náklady na testovanie v neskorších fázach vývoja softvéru. Podľa [8] sa ukázalo, že prvotné príčiny 56 % chýb identifikovaných v softvérových projektoch, boli odhalené práve vo fáze tvorby špecifikácií, pretože boli miestami nejasné, nejednoznačné, nesprávne alebo nekompletné. Na obrázku 2.1 sa nachádza graf zachytávajúci rozloženie chýb v softvérových projektov.



Obr. 2.1: Graf zachytávajúci rozloženie chýb v jednotlivých softvérových projektoch podľa [8].

Podľa [12] 100 % pokrytím špecifikácií dosiahneme pokrytie približne 70 % zdrojového kódu v implementácii. Štúdie, ktoré sa venujú dôležitosti špecifikácií požiadaviek a všeobecne tejto doméne, je viac. V predošlých sekciách sme narazili na skutočnosť, že tvorba naozaj kvalitnej špecifikácie sa v praxi realizuje málokedy, hlavne takej, ktorá vznikne ešte pred implementáciou softvéru. Často sa vytvára až po úvodnej fáze vývoja softvéru alebo nevznikne vôbec. Všeobecne platí, že vytvorením kvalitnej špecifikácie pred samotnou implementáciou sa ušetrí nielen množstvo času, ale aj nákladov na testovanie a podporu softvéru.

2.4 Techniky testovania na základe požiadaviek

Ako sme už vyššie spomenuli, táto doména testovania je tvorená 2 fázami – procesom skvalitnenia špecifikácií a tvorba testovacích prípadov na ich pokrytie. V tejto sekcii popíšeme 2 techniky, ktoré sa venujú týmto 2 fázam.

2.4.1 Recenzia nejednoznačnosti

Ambiguity review (voľne preložené ako *recenzia nejednoznačnosti*) je technika statického testovania, ktorá sa zameriava na identifikáciu a opravenie tých častí špecifikácií, ktoré sú nejasné, nejednoznačné, neúplné alebo logicky nekonzistentné [1]. Možno ju použiť paralelne

už pri tvorbe špecifikácií. Táto technika sa snaží obmedziť propagovanie chýb do neskorších fáz vývoja softvéru. Proces tejto techniky sa uskutočňuje v 2 krokoch:

1. Recenzia človeka, ktorý nie je expertom v doméne – sústreďuje sa na identifikáciu rôznych logických a štrukturálnych nejasností alebo nejednoznačností. Kedže recenzent nepozná podrobne danú doménu, dokáže poukázať na rozličné nejasnosti, ktoré nie sú explicitne obsiahnuté.

2. Recencia experta v danej doméne, ktorý preskúma dokument so zameraním na jeho obsah.

Technika recenzie nejednoznačnosti definuje 15 typov často vyskytujúcich sa chýb v špecifikačnom dokumente [1]. Ako príklad si uvedieme chybu, ktorá sa označuje pod pojmom *dangling else*, ktorá sa môže vyskytnú pri používaní nasledujúcich slov – *musieť*, *bude*, *je jedným z*, *mal by*, *mohol by* alebo *môže byť*. Príklad požiadavky, ktorá obsahuje jedno z uvedených slov:

> Rodinný stav *musí byť* buď ženatý/vydatá, slobodný(á) alebo rozvedený(á).

Táto požiadavka definuje, čo sa stane za normálnych okolností, ale nešpecifikuje, čo sa stane v prípade, ak je rodinný stav napríklad „ovdovelý“ t.j. nešpecifikuje, čo sa stane, ak požiadavka nie je splnená.

Výhodou tejto techniky testovania špecifikácií sú veľmi nízke náklady. Na základe skúseností sa zistilo, že ak sa neodstránia chyby, ktoré sú v špecifikácii pred návrhom a implementáciou, takmer 100 % z nich sa následne objaví v zdrojových kódoch softvéru [2].

2.4.2 Graf príčin a dôsledkov

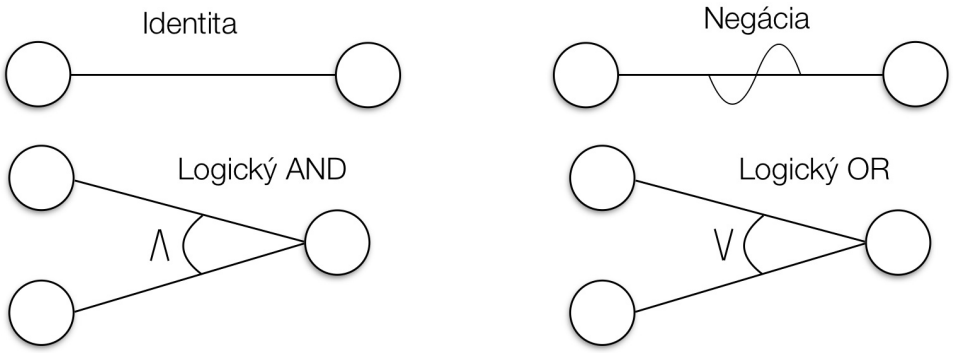
Technika testovania čiernej skrinky za použitia grafu príčin a dôsledkov (angl. *cause-effect graph*). Je to technika tvorby testovacích prípadov podľa grafickej/tabuľkovej metódy, ktorá sa odvodí priamo zo špecifikácie požiadavkov. Testovanie začína *rozdelením špecifikácii na jednotlivé časti* – časťami sa myslí niečo, čo súvisí s činnosťou alebo pozorovateľným výsledkom. Dôvodom rozdelenia špecifikácii je, že rozsiahle požiadavky sú pre testovanie neprehľadné. Po úspešnom rozdelení nasleduje *identifikácia príčin a dôsledkov*, kde:

príčina (angl. *cause*) je vstupná podmienka (vstup systému) alebo ekvivalenčná trieda vstupných podmienok a

dôsledok (angl. *effect*) je výstupná podmienka alebo zmena stavu systému, ktorá musí byť pozorovateľná.

Akonáhle sú všetky príčiny a dôsledky identifikované, označíme ich jednoznačnými identifikátormi napríklad ich očíslujeme. Nasleduje *tvorba grafu príčin a dôsledkov* – sémantický kontext RS sa analyzuje a transformuje do booleovského grafu, kde uzly predstavujú pravdivostnú hodnotu a hrany logickú väzbu. Graf je implicitne orientovaný zľava doprava, pričom najľavejšie uzly sú príčiny a najpravejšie dôsledky. Jednotlivé uzly sú spojené logickými operátormi. V grafu príčin a dôsledkov existujú 4 základné symboly, ktoré sa nachádzajú na obrázku 2.2. Po konštrukcii grafu môžeme zaviesť obmedzenia (angl. *constraints*) nad kombináciami príčin a dôsledkov – môže existovať taký vzťah príčina – dôsledok, ktorý vylučuje niektoré prípady.

Obmedzenie príčin definuje, ktoré kombinácie príčin sú validné. Patria sem:

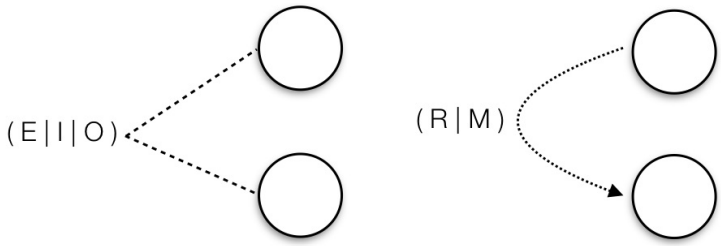


Obr. 2.2: Základné symboly grafu príčin a dôsledkov. **Identita** 2 uzlov znamená, že prvý uzol *spôsobuje* uzol druhý, **negácia** predstavuje *potlačenie* jedného uzla druhým. Logický operátor **AND** znamená prvý `a` druhý uzol *spôsobuje* uzol tretí a logický operátor **OR** zachytáva vzťah prvý `alebo` druhý uzol *spôsobuje* tretí.

- **exclusive (E)** – 0 alebo 1 z uzlov platí (nesmie platiť viac uzlov dohromady),
- **Inclusive (I)** – aspoň 1 z uzlov platí (nesmie byť 0),
- **One (O)** – vždy je platný práve 1 z uzlov,
- **Requires (R)** – musí byť platný druhý uzol, aby prvý mohol platiť.

Obmedzenie dôsledkov definuje prioritu dôsledkov – špeciálnou hranou zabraňujeme vznik ďalšieho uzla, čím sa zvýši aj prehľadnosť grafu. Patrí sem:

- **Masks (M)** – ak prvý uzol platí, druhý je neplatný.



Obr. 2.3: Obmedzenia príčin aj dôsledkov.

Po zostrojení príslušného grafu so zavedenými obmedzeniami príčin a dôsledkov môžeme pristúpiť k *tvorbe rozhodovacej tabuľky*. Realizuje sa to vhodnými metódami [12] a výsledkom je rozhodovacia tabuľka, z ktorej dostaneme jednotlivé testovacie prípady na pokrytie špecifikácie.

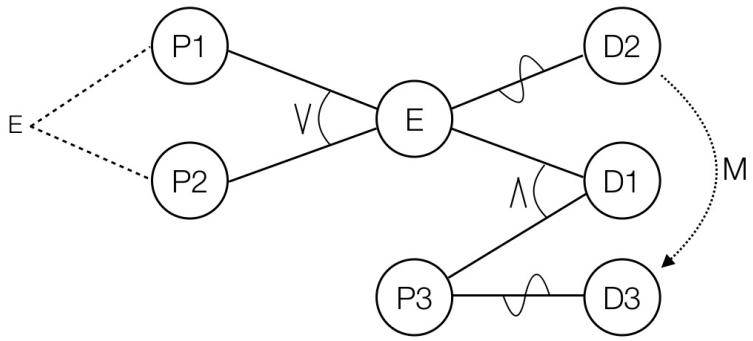
Pre lepšiu demonštráciu tejto techniky uvediem príklad špecifikácie [7], podľa nej vytvorený graf príčin a dôsledkov a odvodenú rozhodovaciu tabuľku.

> Prvý stĺpec obsahuje znak 'P' alebo 'Q'. Druhý stĺpec obsahuje čislo. V takom prípade sa aktualizuje súbor. Ak je prvý znak neplatný, vypíše sa správa A12. Ak druhý stĺpec neobsahuje číslo, vypíše sa správa A13.

Identifikované príčiny a dôsledky:

- `P1` prvý stĺpec obsahuje znak “P”,
- `P2` prvý stĺpec obsahuje znak “Q”,
- `P3` druhý stĺpec obsahuje číslo,
- `D1` aktualizuje sa súbor,
- `D2` vypíše sa správa A12,
- `D3` vypíše sa správa A13,
- `E` explicitný uzol (spojka OR).

Graf príčin a dôsledkov odvodený z vyššie uvedenej špecifikácie a doplnený o obmedzenia príčin a dôsledkov sa nachádza na obrázku 2.4. Použitím vhodných metód [12] sa graf transformuje na rozhodovaciu tabuľku, ktorá nám definuje jednotlivé testovacie prípady. Výslednú rozhodovaciu tabuľku zachytáva tabuľka 2.1.



Obr. 2.4: Graf príčin a dôsledkov odvodený zo špecifikácie a doplnený o odmedzenia.

		T1	**T2**	**T3**	**T4**	**T5**
Príčiny	**P1**: Prvý stĺpec obsahuje znak "P"	F	F	F	T	T
	P2: Prvý stĺpec obsahuje znak "Q"	F	F	T	F	F
	P3: Druhý stĺpec obsahuje číslo	F	T	F	F	T
Dôsledky	**D1**: Aktualizuje sa súbor	F	F	F	F	T
	D2: Vypíše sa správa A12	T	T	F	F	F
	D3: Vypíše sa správa A13	F	F	T	T	F

Tabuľka 2.1: Výsledná rozhodovacia tabuľka.

Kapitola 3

Návrh podporného nástroja pre testovanie založené na požiadavkách

V tejto kapitole sa zoznámime s návrhom pomocného nástroja pre testovanie založené na požiadavkách (ďalej len nástroja). Špecifikácia požiadaviek je uvedený v sekcii 3.2. Architektúra nástroja a podobrobný popis jeho komponentov sa nachádza v sekcii 3.3. Sekcia 3.4 sa zaoberá popisom funkcionality navrhnutého nástroja a v sekcii 3.5 je predstavený návrh užívateľského rozhrania nástroja.

3.1 Definícia pojmov

V tejto sekcii definujeme niektoré dôležité pojmy, ktoré sa budú ďalej v práci používať.

API (Application Programming Interface) označuje rozhranie pre programovanie aplikácií. Ide o zbierku procedúr, funkcií, tried či protokolov nejakej knižnice (taktiež i iného programu alebo jadra operačného systému), ktoré môže programátor využívať. API určuje, akým spôsobom sú funkcie knižnice volané zo zdrojového kódu programu.

REST (Representational State Transfer) je architektúra API, ktorá nám umožňuje pristupovať k dátam a vykonávať nad nimi napríklad CRUD (create, read, update, delete) operácie. REST je bezstavový, čím zjednodušuje komunikáciu s API a umožňuje paralelné spracovanie obsahu.

3.2 Špecifikácia požiadaviek

Cieľom práce je vytvoriť podporný nástroj pre testovanie založené na požiadavkách. Navrhnutý nástroj bude spĺňať nasledujúce požiadavky:

- Implementácia webového rozhrania.
- Intuitívne užívateľské rozhranie.
- Spoľahlivosť funkcionality a konzistencia užívateľského rozhrania vo vybraných webových prehliadačoch – Google Chrome, Mozilla Firefox a Safari.

- Vytváranie a odstraňovanie projektov.
- Vytváranie a odstraňovanie testovacích prípadov.
- Importovanie a exportovanie projektov.
- Uloženie a načítanie projektov.
- Prehľadávanie existujúcich projektov.
- Prepínanie medzi existujúcimi projektami.
- Nahranie a načítanie textového špecifikačného dokumentu.
- Vyznačenie častí dokumentu a priradenie ich k jednotlivým testovacím prípadom.
- Vytvorenie reportu o aktuálnom pokrytí špecifikácie požiadaviek jednotlivými testovacími prípadmi.
- Export reportu do HTML.
- Asynchrónna komunikácia s agentom prostredníctvom navrhnutého REST API.

Reporty budú spĺňať nasledujúce požiadavky:

- Obsahujú informácie o aktuálnom projekte a metriky o výsledkoch testovania.
- Obsahujú špecifikácie požiadaviek.
- Obsahujú užívateľom špecifikované popisy jednotlivých testovacích prípadov.
- Obsahujú odkazy na požiadavky spojené s testovacími prípadmi.

Súčasťou nástroja je agent, ktorý bude spĺňať tieto požiadavky:

- Asynchrónna komunikácia s klientom prostredníctvom navrhnutého REST API.
- Vytváranie testovacích šablón.
- Škálovateľnosť z pohľadu programovacích jazykov a ich testovacích frameworkov.
- Spúšťanie existujúcich testovacích sád a spracovanie ich výsledkov.
- Vrátenie spracovaných výsledkov testovania klientskej časti nástroja.
- Možnosť ukladania a načítania projektov z klientskej časti nástroja.

3.3 Architektúra nástroja

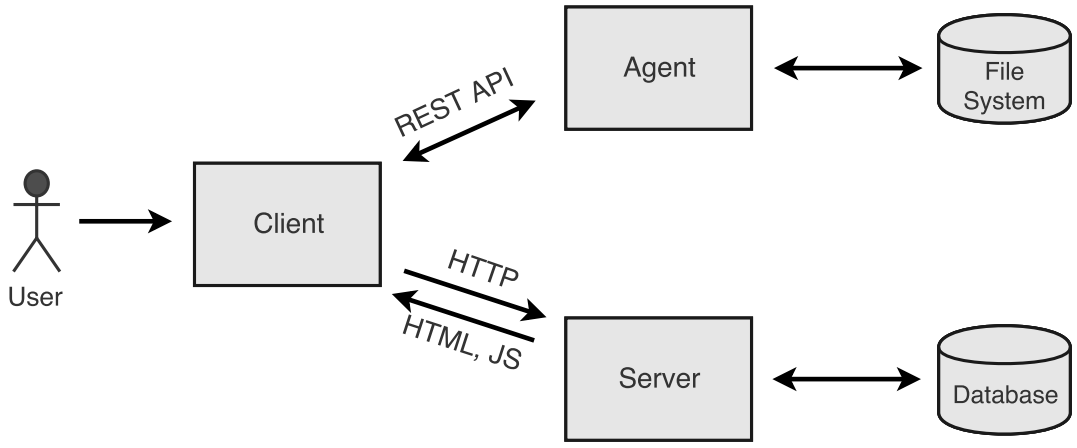
Táto sekcia sa venuje popisu architektúry navrhnutého nástroja. Jednou z požiadaviek nástroja je implementácia webového rozhrania, čiže sa požaduje, že daný nástroj bude dostupný pre užívateľa prostredníctvom webového prehliadača. Na to, aby bola táto požiadavka splnená, je potrebné, aby jadrom nástroja bol webový server. Výhodou webového rozhrania je všadeprítomnosť webového prehliadača ako klienta a možnosť používania nástroja bez nutnosti jej inštalácie.

Na obrázku 3.1 sa nachádza základná schéma architektúry navrhnutého nástroja. Nástroj je rozdelený na 3 časti:

- **klient** – implementuje užívateľské rozhranie, prostredníctvom ktorého užívateľ interaguje s nástrojom,
- **server** – poskytuje klientsku časť nástroja a zaisťuje perzistenciu dát,
- **agent** – komunikuje s klientskou časťou nástroja, generuje testovacie šablóny a spúšťa testovacie sady.

pričom klient a server spolu vytvárajú webovú aplikáciu. Prítomnosť agenta spočíva v realizácii autonómneho vykonávania tej časti funkcionality, ktorá sa stará o vytváranie testovacích šablón, spúšťanie testovacích sád a spracovanie výsledkov testovania. Je to vhodné pre zaistenie plynulosti práce užívateľa s nástrojom. Užívateľ nie je obmedzovaný čakaním na niektoré zo spomenutých udalostí a môže bez komplikácií pokračovať ďalej v interakcii s nástrojom.

Návrh architektúry a popis jednotlivých komponentov webovej aplikácie je podrobne predstavený v sekcii 3.3.1. Ďalej sekcia 3.3.2 sa venuje návrhu a popisu agenta. Klientska časť webovej aplikácie a agent spolu komunikujú prostredníctvom REST API, ktorému sa venuje sekcia 3.3.3.



Obr. 3.1: Architektúra pomocného nástroja pre testovanie založené na požiadavkách.

3.3.1 Architektúra webovej aplikácie

V špecifikácii požiadaviek nástroja sa nachádza niekoľko bodov týkajúce sa správy projektov a testovacích prípadov. Konkrétne sa týkajú týchto požiadaviek:

- Vytváranie a odstraňovanie projektov.
- Importovanie a exportovanie projektov.
- Uloženie a načítanie projektov.
- Prehľadávanie existujúcich projektov.
- Prepínanie medzi existujúcimi projektami.
- Nahranie a načítanie textového špecifikačného dokumentu.

Dôsledkom špecifikácie požiadaviek je perzistencia dát na serverovej časti. Jedným z možností zaistenia perzistencie dát je databáza. Návrh dátovej časti webovej aplikácie sa nachádza v podsekcii 3.3.1.

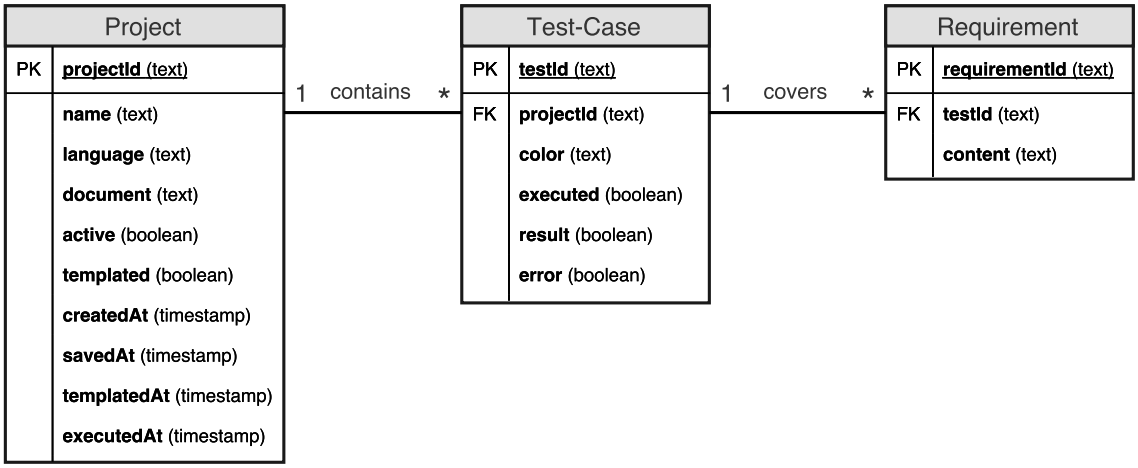
Webová aplikácia teda musí poskytovať užívateľovi potrebné interaktívne užívateľské rozhranie vo forme webovej stránky, ku ktorému má prístup prostredníctvom webového prehliadača. Ďalej musí zabezpečiť potrebnú funkcionalitu, ktorá vyplýva zo špecifikácie požiadaviek. Napokon musí realizovať komunikáciu s databázou, ktorá bude reflektovať aktuálny stav na serveri. Tieto požiadavky sa dajú vhodne namapovať na tzv. *trojvrstvovú architektúru*, ktorá je najznámejším variantom viacvrstvových architektúr informačných systémov a webových aplikácií [6].

Trojvrstvová architektúra sa skladá:

- z **prezentačnej vrstvy**, ktorá vizualizuje infomácie pre užívateľa vo forme webovej stránky prostredníctvom webového prehliadača,
- z **aplikačnej vrstvy** (tiež **business logic**) tvoriaca jadro aplikácie, následne logiku a funkcie, spracovanie dát a operácie nad databázou,
- z **dátovej vrstvy**, ktorú tvorí databáza.

Dátová časť

Ako bolo vyššie spomenuté, dôsledkom špecifikácie požiadaviek je zaistenie perzistencie dát. Pri návrhu sa zohľadnila voľba databázy ako vhodného prostriedku na zaistenie perzistencie dát. V databáze potrebujeme mať záznamy o vytvorených projektoch. Každý projekt môže mať niekoľko testovacích prípadov a každý testovací prípad môže pokrývať niekoľko požiadaviek zo špecifikácii požiadaviek pre daný projekt. Na vyjadrenie patričných vzťahov medzi jednotlivými záznamami použijeme relačnú databázu [14]. Schéma navrhnutej relačnej databázy sa nachádza na obrázku 3.2 vo forme ERD diagramu.



Obr. 3.2: ERD diagram navrhnutej relačnej databázy pre vytvorené projekty, testovacie prípady a požiadavky, ktoré sú nimi pokryté.

Nasleduje popis atribútov relácií:

Project

- **projectId** - primárny kľúč, jednoznačný identifikátor projektu,
- **name** - názov projektu,
- **language** - programovací jazyk, v ktorom sa budú generovať testovacie šablóny,
- **document** - špecifikácia požiadaviek vo forme textového dokumentu,
- **active** - príznak určujúci aktívny projekt t.j. projekt, s ktorým užívateľ pracuje,
- **templated** - príznak určujúci, či pre daný projekt existuje vygenerovaná testovacia šablóna,
- **executed** - príznak určujúci, či bola testovacia sada už vykonaná,
- ∗**At** - všetky atribúty, ktorých názov končí *At* udávajú časovú pečiatku udalostí.

Test-Case

- **testId** - primárny kľúč, jednoznačný identifikátor testovacieho prípadu,
- **projectId** - cudzí kľúč odkazujúci sa na primárny kľúč relácie *Project*,
- **color** - farba v hexadecimálnom tvare, ktorá predstavuje daný testovací prípad,
- **result** - príznak reprezentujúci výsledok testovacieho prípadu,
- **error** - príznak určujúci správne vykonanie testovacieho prípadu.

Requirement

- **requirementId** - primárny kľúč, jednoznačný identifikátor pokrytej požiadavky,
- **testId** - cudzí kľúč odkazujúci sa na primárny kľúč relácie *Test-Case*,
- **content** - obsah vyznačenej požiadavky.

3.3.2 Architektúra agenta

Na rozdiel od webovej aplikácie, architektúra agenta je jednoduchšia. Jedinou požiadavkou agenta je existencia prístupových bodov REST API a poskytnutie požadovaných služieb klientskej časti nástroja. Agent naslúcha na vyhradenom porte, prijíma požiadavky od klienta a odpovedá na ne podľa navrhnutého REST API.

3.3.3 Popis REST API

Táto sekcia sa venuje popisu aplikačného rozhrania medzi klientom a agentom. V tejto sekcii sa nachádza všeobecný popis jednotlivých prístupových bodov REST API. V prílohe B sa nachádza úplný popis REST API s uvedenými parametrami klienta a odpoveďami agenta, ktoré sú prispôsobené implementácii. Pri popise REST API sme sa inšpirovali zo stránky popisu REST API sociálnej siete Twitter[1].

GET /create – vytvorenie projektu na strane agenta. Agent vytvorí úložisko pre daný projekt, ktorý bude využívať pri obsluhe ďalších požiadaviek klienta.

GET /delete – odstránie projektu na strane agenta. Agent odstráni všetky závislosti daného projektu.

POST /save – uloženie aktuálneho stavu projektu a jeho testovacích prípadov na strane agenta.

[1]Popis REST API sociálnej siete Twitter – https://dev.twitter.com/rest/public

GET /savedProjects – získanie poľa názvov všetkých uložených projektov na strane agenta.

GET /load – získanie uloženého stavu projektu a jeho testovacích prípadov na strane agenta.

POST /template – vytvorenie testovacej šablóny na strane agenta. Tvorba testovacej šablóny sa realizuje v programovacom jazyku, ktorý užívateľ špecifikuje pri vytvorení projektu na strane serverovej aplikácie, resp. v klientskej časti.

GET /execute – spustenie testovacej sady na strane agenta a návrat výsledkov testovania.

3.4 Funkcionalita nástroja

Nasleduje popis jednotlivých častí funkcionality nástroja na základe požiadaviek.

3.4.1 Funkcionalita webovej aplikácie

V tejto podsekcii si popíšeme funkcionalitu webovej aplikácie. Funkcionalitu webovej aplikácie tvorí:

1. tvorba a správa projektov,
2. tvorba testovacích prípadov,
3. práca so špecifikačným dokumentom,
4. generovanie testovacej šablóny, testovanie a vizualizácia výsledkov,
5. tvorba reportu a export do HTML.

Tvorba a správa projektov

Server by mal umožniť užívateľovi vytvoriť projekt 3 spôsobmi:

- vytvoriť nový projekt,
- importovať projekt zo súborového systému, ktorý bol v minulosti úspešne exportovaný,
- načítať projekt, ktorý je uložený na súborovom systéme, ku ktorému pristupuje agent.

Posledná možnosť by mala byť uskutočnená asynchrónnym prenosom medzi klientom a agentom prostredníctvom navrhnutého REST API. Klient musí mať možnosť existujúce projekty aj odstraňovať či už iba z databáze alebo aj na strane agenta.

Ďalšími požiadavkami v doméne správy projektov je už spomínané exportovanie aktuálneho stavu projektov, ich uloženie na strane agenta a možnosť prehľadávať a zobrazovať existujúce projekty. Dôležitým detailom tejto domény je vizualizácia názvu aktuálneho projektu, s ktorým užívateľ pracuje.

Tvorba testovacích prípadov

Táto časť funkcionality je tvorená z vytvárania a odraňovania testovacích prípadov, ich vizualizácia v užívateľskom rozhraní. Spôsob priradenia požiadaviek z dokumentu testovacím prípadom je popísaný v nasledujúcej podsekcii. Je dôležité dosiahnuť, aby užívateľ jednoznačne rozlíšil, ktoré požiadavky pokrýva ktorý testovací prípad. To sa realizuje pomocou farby, ktorá sa vygeneruje náhodne pri vytvorení testovacieho prípadu. Touto farbou budú označené tie požiadavky, ktoré sú priradené k danému testovaciemu prípadu.

Práca so špecifikačným dokumentom

Je to jedna z najdôležitejších častí funkcionality nástroja. V prvom rade musí byť umožnené užívateľovi nahrať vybraný textový dokument obsahujúci špecifikácie požiadaviek na server, ktorý ich správne zobrazí v užívateľskom rozhraní. Až po realizácii tejto udalosti môže užívateľ pristúpiť k priraďovaniu požiadaviek k testovacím prípadom. Samotné priradenie požiadavky (všeobecne ľubovoľnej časti dokumentu) sa vykoná jednoduchým vyznačením pomocou kurzoru myši. Po vyznačení požiadavky, server zaistí jej priradenie k vyznačenému (aktívnemu) testovaciemu prípadu, resp. uloží vyznačenú požiadavku do atribútu *content* relácie *Requirement*, ktorej inštancia sa pridá do atribútu *requirements* (je datového typu pole) relácie *Test-Case*.

Generovanie testovacej šablóny, testovanie a vizualizácia výsledkov

Táto časť funkcionality je realizovaná agentom. Klient mu posiela dáta vo formáte, ktorý požaduje REST API (popísané v podsekcii 3.3.3). Agent spracuje požiadavky klienta a vráti požadované dáta. Pri požiadavke o vygenerovaní testovacej šablóny agent vráti príznak, či sa generovanie vykonalo úspešne. Iné je to pri samotnom spúšťaní testovacej sady a spracovaní výsledkov. Klient dostane v odpovedi serializované dáta o výsledku testovania. Tieto dáta obsahujú výsledky všetkých testovacích prípadov, čiže klient na základe týchto informácií označí každú vyžnačenú požiadavku príslušnou farbou. Zelenou označí tie požiadavky, ktorých testovacie prípady vrátili *PASS*. Naopak červenou farbou sa označia tie požiadavky, ktorých testovacie prípady vrátili *FAIL*.

Tvorba reportu a export do HTML

Vytvorenie reportu je výstupom tohto nástroja. Report obsahuje všetky údaje o danom projekte, jeho testovacích prípadoch a o aktuálnom pokrytí špecifikácii požiadaviek projektu. Ide o serializáciu dát konkrétneho projektu na serveri a zobrazenie v klientskej časti. Exportovanie do HTML už dokážu zabezpečiť dnešné webové prehliadače, dokonca užívateľ má možnosť daný report i stiahnuť vo formáte PDF.

3.4.2 Funkcionalita agenta

V tejto podsekcii sa budeme venovať funkcionalite agenta. Funkcionalitu agenta tvorí:

1. tvorba testovacích šablón,
2. spúšťanie testovacích sád,
3. uloženie a načítanie projektov.

Tvorba testovacích šablón

Agent zachytí požiadavku `POST /template` s príslušnými parametrami – *projectId*, *projectName*, *projectLanguage* a *tests*. Na základe informácie o programovacom jazyku určí, ktorú metódu zavolá na generovanie testovacej šablóny. Vybraná metóda sa zavolá s parametrami *projectName*, ktorý obsahuje názov projektu a *tests*, ktorý obsahuje pole objektov typu testovací prípad. Použitím týchto parametrov daná metóda vygeneruje šablónu testovacej sady vo vybranom jazyku. Testovacia sada bude mať tieto vlastnosti:

- bude obsahovať toľko testovacích metód, koľko je objektov typu testovací prípad v poli *tests*,
- v komentároch testovacích metód budú prítomné jednotlivé požiadavky, ktoré boli priradené konkrétnym testovacím prípadom.

Agent pošle odpoveď klientovi na základe úspešnosti vygenerovania testovacej sady.

Príklad vygenerovanej testovacej šablóny v programovacom jazyku Python:

```
# !/usr/bin/env python
# -*- coding: utf-8 -*-

import unittest

class TestNazovProjektu(unittest.TestCase):

    def test_nazov_testovacieho_pripadu(self):
        """Selected requirements in document:
        1 'Zoznam vyznačených požiadaviek z~dokumentu'
        """
        # write your code here
        self.assertTrue(False)

if __name__ == '__main__':
    unittest.main()
```

Spúšťanie testovacích sád

Po úspešnom vygenerovaní testovacej šablóny musí užívateľ dopísať telá testovacích metód. Vzhľadom na to, že táto práca neobsahuje v špecifikácii požiadaviek spôsob doplnenia testovacej šablóny o telá testovacích metód a dodanie potrebnej SUT, je táto činnosť prenechaná na samotného užívateľa.

Agent obdrží požiadavku `GET /execute` s jediným parametrom – *projectName*, podľa ktorého je schopný nájsť cestu k súboru s testovacou sadou a podľa zakončenia súboru ho spustí s príslušnou konfiguráciou. Výsledok testovania sa získava parsovaním štandardného výstupu resp. štandardného chybového výstupu. Informácie o výsledku testovania sa serializujú a pošlú späť klientovi.

Uloženie a načítanie projektov

Uloženie projektu realizuje požiadavka `POST /save`, pri ktorom klient pošle všetky dáta o projekte v serializovanom formáte, agent ich obdrží a vloží ich buď do vlastnej databázy

alebo na súborový systém.

Načítanie projektu prebieha v dvoch fázach:

- agent najprv obdrží požiadavku `GET /savedProjects`, na ktorú odpovedá poľom názvov všetkých uložených projektov,
- až potom obdrží požiadavku `GET /load` s parametrom *projectName* obsahujúcim názov uloženého projektu, ktorý chce klient načítať.

3.5 Uživateľské rozhranie nástroja

Užívateľmi tohto nástroja budú testeri. Ide o náročnejších užívateľov, ktorí sú zdatní v oblasti informačných technológií a pre svoju prácu využívajú veľa rôznych nástrojov a programov. Prvá interakcia testera so vzniknutým nástrojom bude kľúčová z hľadiska životnosti. Užívateľské rozhranie (ďalej len UI) navrhnutého nástroja musí byť jednoduché, účinné a spoľahlivé. V opačnom prípade hrozí nezáujem používania nástroja. Návrh UI odpovedá návrhu klientskej časti webovej aplikácie, s ktorým užívateľ interaguje. Agent nemá UI, má iba navrhnuté rozhranie v podobe REST API.

Pri návrhu UI musíme splniť nasledujúce požiadavky:

- intuitívne UI,
- spoľahlivosť funkcionality a konzistencia UI v rozličných webových prehliadačoch (Google Chrome, Mozilla Firefox, Safari).

V tejto sekcii sa nachádza popis návrhu jednotlivých častí UI. Celé UI je rozdelené na podčasti podľa funkcionality webovej aplikácie. Nasleduje popis jednotlivých podčastí UI.

Navigácia

Navigačné menu musí obsahovať všetky možnosti, ktoré súvisia s projektom, s ktorým užívateľ momentálne pracuje. Sú to voľby:

- vytvoriť a odstrániť projekt,
- importovať a exportovať projekt,
- uložiť a načítať projekt,
- prehľadávať existujúce projekty,
- zobraziť iný projekt.

Na obrázku 3.3 sa nachádza drôtený model zachytávajúci vzhľad navigačného menu. Navigácia je navrhnutá ako tlačidlo, po kliknutí ktorého sa zobrazí ponuka (angl. *drop-down button*) s uvedenými činnosťami. Horná časť ponuky obsahuje zoznam činností vzťahujúcich sa na aktívny projekt a dolná časť obsahuje zoznam ostatných neaktívnych projektov. Po kliknutí na ľubovoľný z nich klient zaistí zobrazenie vybraného projektu.

Dokument špecifikácie požiadaviek

Znázornenie dokumentu so špecifikáciami požiadaviek je uvedené na obrázku 3.4. Nachádza sa na pravej časti drôteného modelu. Nevyhnutnou súčasťou zobrazenia dokumentu je tlačidlo *Browse*, pomocou ktorého užívateľ realizuje nahranie dokumentu do nástroja.

Obr. 3.3: Drôtený model znázorňujúci návrh navigačného tlačidla nástroja.

Tabuľka s testovacími prípadmi

Prehľadný spôsob zobrazenia testovacích prípadov je formou tabuľky. Každý riadok tabuľky obsahuje meno testovacieho prípadu, jeho farbu, ktorou budú označované časti požiadaviek v dokumente, informácia o výsledku testovania a tlačidlo na odstránenie testovacieho prípadu. V dolnej časti tejto tabuľky sa nachádzajú tri tlačidlá – na vytvorenie testovacieho prípadu, na vygenerovanie testovacej šablóny a na spustenie testovacej sady. Na obrázku 3.4 je znázornená táto tabuľka na ľavej časti drôteného modelu.

Report o aktuálnom pokrytí špecifikácie požiadaviek

Pri návrhu reportu musíme splniť jednotlivé body špecifikácie požiadaviek týkajúce sa reportov. Každý report obsahuje:

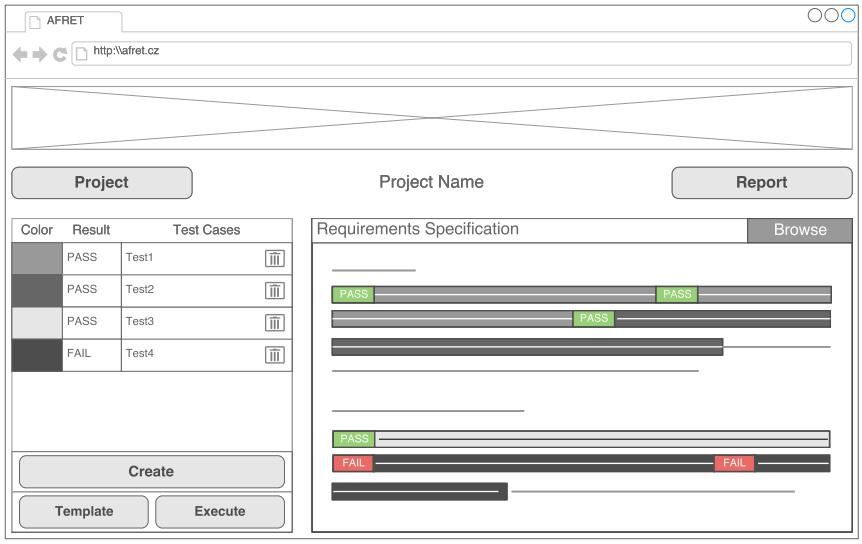
- informácie o aktuálnom projekte a metriky o výsledkoch prebehnutých testov,
- špecifikácie požiadaviek,
- užívateľom špecifikované popisy jednotlivých testovacích prípadov,
- odkazy na požiadavky spojené s testovacími prípadmi.

Na obrázku 3.5 sa nachádza návrh stránky s reportom. Informácie o aktuálnom projekte a metriky o výsledkoch testovania sa nachádza na hornej časti drôteného modelu, nižšie je umiestnené zobrazenie špecifikácie požiadaviek s vyznačenými časťami príslušnými farbami podľa testovacích prípadov. V dolnej časti drôteného modelu sa nachádza zoznam testovacích prípadov s odkazmi na požiadavky spojené s nimi a každý testovací prípad obsahuje užívateľom špecifikovaný popis vo forme názvu testovacieho prípadu.

Výsledný návrh

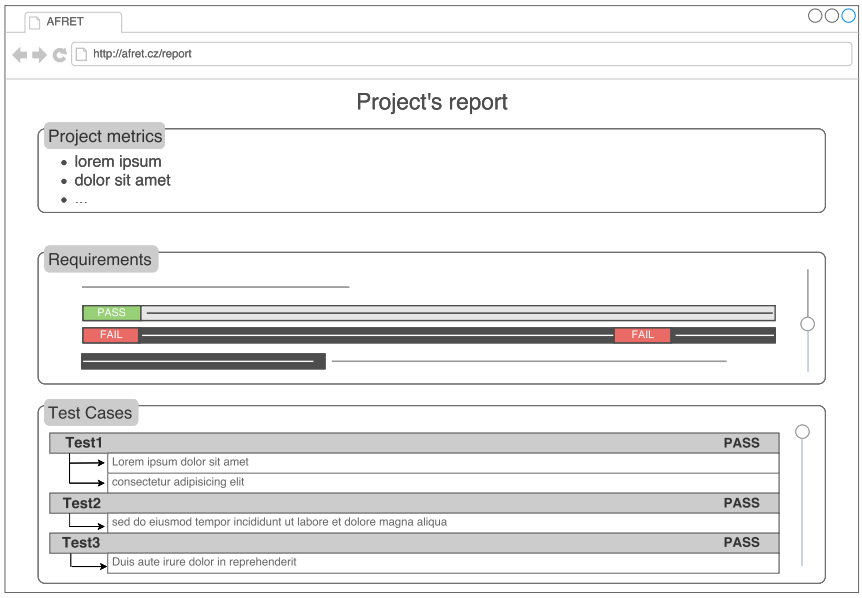
Na obrázku 3.4 sa nachádza drôtený model, ktorý zachytáva výsledný vzhľad navrhnutého UI. Výslednú aplikáciu tvorí jedno okno, ktoré obsahuje tieto podčasti:

- navigačné tlačidlo *Project* s voľbami správy projektu,



Obr. 3.4: Drôtený model znázorňujúci výsledný vzhľad UI.

- tabuľka s testovacími prípadmi a tlačidlami na vytvorenie testovacieho prípadu *Create*, na generovanie testovacej šablóny *Template* a na spustenie testovacej sady *Execute*,
- element, ktorý zobrazí nahratý textový špecifikačný dokument a tlačidlo *Browse*, prostredníctvom ktorého sa realizuje nahranie dokumentu,
- tlačidlo *Report*, ktoré v novom okne zobrazí prehľadný report o aktuálnom pokrytí špecifikácie požiadavkov.



Obr. 3.5: Drôtený model znázorňujúci návrh reportu o aktuálnom pokrytí špecifikácií požiadaviek.

Kapitola 4

Implementácia nástroja pre testovanie založené na požiadavkách

V tejto kapitole popíšeme implementáciu pomocného nástroja pre testovanie založené na požiadavkách. V prvej sekcii 4.1 predstavíme jednotlivé technológie, ktoré sa použili pri implementácii. Ďalej v sekcii 4.2 popíšeme implementáciu užívateľského rozhrania – ktoré technológie sa pri tom použili a ako sa uskutočňuje zobrazenie dát z databázy v UI. Implementáciou funkcionality sa venuje sekcia 4.3.

4.1 Výber technológií

Podľa špecifikácie požiadaviek, ktoré obsahujú požiadavku na implementáciu webového rozhrania, je jasné, že výber technológií sa uskutočnil v oblasti webových technológií.

4.1.1 HTML, CSS

Každá webová stránka sa zakladá aspoň na minimálnom množstve kóde značkovacieho jazyka HTML. CSS je jazyk pre popis spôsobu zobrazenia elementov na webových stránkach napísaných v jazykoch HTML, XHTML alebo XML [5]. Tieto jazyky tvoria základ dnešných webových stránok.

4.1.2 Javascript

Javascript [13] je skriptovací, prototypovo založený, objektovo-orientovaný programovací jazyk. Kód v jazyku Javascript sa dá vložiť priamo do HTML dokumentov, pretože webové prehliadače obsahujú interpret tohto jazyka. Často sa používa na obsluhu udalostí a užívateľských akcií (stisknutie tlačidla atď.), ovládanie prehliadača, dynamickú zmenu obsahu dokumentu pomocou manipulácie s DOM alebo na asynchrónnu komunikáciu so serverom pomocou technológie AJAX. Nepríjemnou skutočnosťou na interpretoch jazyka Javascript vo webových prehliadačoch je, že nie vždy spĺňajú špecifikáciu jazyka a stáva sa, že rôzne interprety vykonajú ten istý kód s odlišným výsledkom. Programátor je potom nútený na toto myslieť pri písaní kódu v rozličných webových prehliadačoch.

Dôvodom výberu Javascriptu ako hlavného programovacieho jazyka pre daný nástroj je dohoda s vedúcim práce a dôkladná analýza jeho vlastností a frameworkov. Usúdili sme, že Javascript je vhodný na obsluhu udalostí a všeobecne na vývoj webových aplikácií.

Použité javascriptové frameworky

Meteor.js[1] je vývojová platforma zložená z dvoch základných častí. Prvý z nich je knižnica balíkov `packages` – samostatné moduly, ktoré vykonávajú rôzne funkcie ako odosielanie emailu, prihlasovanie, používanie šablón atď. Balíky, ktoré sú vytvorené komunitou sa dajú získať z Atmosphere[2]. Druhou časťou Meteoru je nástroj pre príkazový riadok `meteor`. Tento príkaz zaisťuje kroky potrebné k spusteniu serverovej aplikácie, inštaláciu nových balíkov a nasadenie do prevádzky. Po zadaní príkazu, Meteor sleduje zmeny v projekte, automaticky kompiluje Javascript, vykonáva šablóny do Javascriptu a zlučuje `.css` súbory. Autori Meteoru spísali 7 hlavných princípov, ktoré tento framework charakterizujú. Všetky sú definované v dokumentácii.

Práca s dátami Odosielať časti HTML alebo celé stránky pri každom dopyte na server zbytočne zaťažujú sieť a spomaľujú prácu. Preto je lepšie posielať v rámci komunikácie iba dáta a nechať klientskú aplikáciu rozhodnúť, ako s nimi naloží.

Jeden jazyk Klientská i serverová časť aplikácie by mali používať rovnaký programovací jazyk, v prípade Meteoru je to Javascript.

Všadeprítomná databáza K databáze sa v Meteore pristupuje tak z klientskej ako aj zo serverovej časti. Dopyty na MongoDB sú zadávané úplne rovnako a vrátia rovnaké výsledky.

Kompenzácia oneskorenia (angl. latency compensation) Meteor okamžite simuluje zmeny vykonané na klientskej časti, ešte predtým, ako sa vráti odpoveď servera. Aplikácia vyvolá dojem, že spojenie s databázou nemá žiadne oneskorenie a všetko sa deje v reálnom čase. Pokiaľ dôjde na servere k chybe pri ukladaní alebo zmene dát, je o správnosti dát na strane klienta postarané, akonáhle sú doručené.

Reaktivita vo všetkých vrstvách Všetky vrstvy aplikácie by mali byť riadené udalosťami.

Využitie ekosystému Vzhľadom na to, že Meteor je open-source a je tiež postavený na open-source technológii (Node.js), je podpora na využitie voľne dostupných nástrojov a frameworkov.

Jednoduchosť = Produktivita Jednoduchý, čistý a prehľadný kód sa číta lepšie, dá sa znovupoužiť, zapúzdriť, využiť ako modul. Preto tvorcovia Meteoru podporujú práve taký design aplikácií.

Dôvodom výberu tohto frameworku bol fakt, že spĺňa požiadavky na implementáciu tohto nástroja. Samozrejme, daný nástroj by sa dal implementovať aj v iných technológiách, ale vzhľadom na spĺňajúce požiadavky a súhlas vedúceho tejto práce sme usúdili, že to bude vhodný výber.

[1] Meteor.js dokumentácia – http://docs.meteor.com/#/full/
[2] Atmosphere.js – https://atmospherejs.com/

Node.js[3] je javascriptová platforma postavená na V8[4], ktorá slúži na tvorbu rýchlych a škálovateľných sieťových aplikácií. Obsahuje, mimo iné, i modul HTTP, ktorý umožňuje vytvoriť webový server. Vďaka webovému serveru v Node.js je možnosť vyvíjať serverovú časť aplikácie v Javascripte, čo prináša celej aplikácii jednotnosť. Webový server v Node.js pracuje iba v jednom vlákne, čo je výrazný rozdiel oproti častejšiemu prístupu, kedy webový server vytvára nové vlákno pre spracovanie každej požiadavky. Aby nedošlo k „zamrznutiu" servera pri čakaní na I/O operáciu, tak sa spracováva každá asynchrónne. Súčasťou nástrojov spojených s Node.js je balíčkovací systém `npm`, ktorý dnes obsahuje viac než 250 000[5] rôznych balíčkov. Jeden balíček zodpovedá jednému znovupoužiteľnému softvérovému projektu či rozšíreniu súvisiaceho s vývojom webových aplikácií.

Express.js[6] je minimalistický webový framework vytvorený v jazyku Javascript. Prináša podporu pre MVC architektúru, routovanie, testovanie a kešovanie (zrýchlenie komunikácie medzi klientom a serverom pomocou vyrovnávacej pamäte). V porovnaní s ostatnými frameworkami, uvedenými v tejto sekcii, je Express veľmi odľahčený. Jeho výhoda spočíva hlavne v rýchlosti a možnostiach prispôsobenia pomocou rôznych modulov. Aby bolo možné na servere spustiť Express, musí byť na ňom nainštalovaný softvér Node.js.

S vedúcim práce sme sa rozhodli celý nástroj implementovať v jazyku Javascript. Vzhľadom na fakt, že nástroj je rozdelený na webovú aplikáciu a agent, serverová časť musí byť postavená na platforme Node.js. Pri implementácii sa použil Node.js verzia 5.7.1 a jeho webový framework *Express.js* verzia 4.13.5.

4.1.3 Databáza MongoDB

MongoDB[7] je multiplatformná dokumentová databáza napísaná v jazyku C++. Radí sa medzi NoSQL databázy a miesto tradičných relačných databáz využívajúcich tabuľky používa dokumenty podobné formátu JSON (MongoDB formát nazývá BSON) a dynamickú databázovú schému, ktorá umožňuje vytváranie a integráciu dát pre aplikácie jednoduchšie a rýchlejšie. Jedná sa o open-source softvér vydaný pod GNU Affero General Public License a Apache licenciami .

4.1.4 jQuery, Bootstrap

jQuery[8] je jedna z najznámejších Javascriptových knižníc na svete. Možno potvrdiť, že vytvára abstraktnú vrstvu nad klientskym Javascriptom. Ponúka funkcie, ktoré výrazne zjednodušujú riešenie problémov, ako je manipulácia s HTML dokumentom, spracovanie udalostí, animácie alebo technológia AJAX. Nespornou výhodou jQuery je aj to, že zatieňuje rôznorodosť implementácie interpretov vo webových prehliadačoch. Kód tak stačí napísať iba raz a jQuery zaistí, aby správne fungoval vo všetkých bežne používaných webových prehliadačoch.

Bootstrap[9] je populárny frontend framework, ktorý umožňuje napríklad veľmi jednoducho vytvoriť mriežkové rozloženie stránky (angl. grid layout), ktoré sa môže dokonca

[3] Node.js dokumentácia – https://nodejs.org/en/docs/
[4] **V8** – interpret jazyka Javascript vyvinutý firmou Google pre ich webový prehliadač *Chrome*
[5] Dostupné z http://www.modulecounts.com/
[6] Express.js dokumentácia – http://expressjs.com/
[7] MongoDB dokumentácia – https://docs.mongodb.com/
[8] jQuery dokumentácia – https://api.jquery.com/
[9] Bootstrap dokumentácia – http://getbootstrap.com/components/

dynamicky prispôsobovať veľkosti zobrazovacieho zariadenia, čo je vhodné pri tvorbe responzívneho designu. Pomáha s typografiou alebo formátovaním tabuliek, formulárov, navigáciou či tlačidiel. Bootstrap taktiež zjednodušuje vytváranie interaktívnych prvkov UI. Obsahuje komponenty, ktoré napríklad výrazne zjednodušujú tvorbu rozbalovacích navigačných panelov, tlačidiel uchovávajúce ich stav, záložiek, vysúvacích prvkov alebo modálnych dialógových okien. Okrem spomínaných výhod je dôležité spomenúť jeho kompatibilitu s najnovšími verziami všetkých bežne používaných webových prehliadačov.

4.1.5 AJAX

AJAX (*Asynchronous Javascript and XML*) je všeobecné označenie pre technológie vývoja interaktívnych webových aplikácií, ktoré menia obsah svojich stránok bez nutnosti ich kompletného znovunačítania[10].

Jedna z požiadaviek na implementáciu nástroja je asynchronná komunikácia medzi klientom a agentom prostredníctvom navrhnutého REST API. Táto požiadavka je implementovaná využitím technológie AJAX.

Použité knižnice

Uvádzame zoznam použitých knižníc a balíčkov, ktoré nám zjedodušili prácu pri implementácii dodaním potrebných funkcií a modulov.

Editable-text[11] implementuje widget, ktorý sa dá namapovať na dokument v Mongo kolekcii, na definíciu udalosti a obsluhu tejto udalosti vo forme jednoduchej možnosti premenovania tohto dokumentu,

Simple-schema[12] dodáva tvorbu databázovej schémy a jej validáciu,

Collection2[13] pridáva možnosť napojenia databázovej schémy na Mongo kolekciu,

Filesaver[14] obsahuje klientsku funkciu `saveAs()`, ktorá uloží dáta do súboru na súborový systém užívateľa,

Toastr[15] funkcia na neblokujúce notifikácie užívateľa,

Router[16] stará sa o routovanie v aplikácii.

4.2 Implementácia užívateľského rozhrania

Táto sekcia popisuje úvodnú fázu implementácie, v ktorej sa naimplementovalo UI podľa návrhu bez funkcionality. Výsledný vzhľad a rozmiestnenie elementov v UI sa naimplelentovalo s využitím jazyka HTML, CSS a frameworku Bootstrap, ktorý bol použitý na mriežkové rozloženie stránky, ktorým sa dosiahla aj responzívnosť stránky na rôzne veľkosti mobilných zariadení, modálne okná, tlačidlá a pod.

[10] AJAX – https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started
[11] Editable-text – https://github.com/jackadams/meteor-editable-text
[12] Simple-schema – https://github.com/aldeed/meteor-simple-schema
[13] Collection2 – https://github.com/aldeed/meteor-collection2
[14] Filesaver – https://github.com/pfafman/meteor-filesaver
[15] Toastr – https://atmospherejs.com/chrismbeckett/toastr
[16] Router – https://github.com/iron-meteor/iron-router

4.2.1 Príprava databázy

Dôležitou súčasťou webovej aplikácie je existencia databáze. V sekcii 4.1 bola predstavená NoSQL databáza MongoDB, ktorá sa použila v rámci nástroja. Súbor `app/lib/collections.js` obsahuje vytvorené schémy databázy s uvedenými dátovými typmi a definovanými implicitnými hodnotami. Pri tvorbe schémy sa použila knižnica *Simple-scheme*. Knižnica *Collection2* implementuje spôsob naviazania schémy na vytvorenú Mongo kolekciu. Tým sa zaistí integrita dát [14].

Keďže jeden z princípov frameworku Meteor.js je *všadeprítomná databáza*, programátor má prístup k databáze na ktoromkoľvek mieste v zdrojových súboroch. Kvôli prehľadnosti zdrojových kódov sa definovali metódy na rôzne operácie nad databázou v súbore `app/lib/methods.js`, ktorý obsahuje špeciálnu metódu triedy Meteor – `Meteor.methods()`, parametrom ktorého je objekt obsahujúci definície metód. Pristupovať k týmto metódam Meteor zaisťuje pomocou metódy `Meteor.call()`.

4.2.2 Zobrazenie dát v užívateľskom rozhraní

V predošlých sekciách sa popísala implementácia užívateľského rozhrania a príprava databázy. Nasleduje popis spôsobu zobrazenia dát z databázy v užívateľskom rozhraní. Meteor používa šablónovací systém *Spacebars* na vytváranie HTML šablón. Tieto šablóny obsahujú statický HTML kód, premenné ako zdroje dát a riadiace konštrukcie definované pomocou dvojitých zložených zátvoriek `{{ }}` napr. `if`, `each`, `with` a pod. Spacebars umožňuje vnorenie týchto šablón, čo umožňuje štrukturovanú tvorbu výsledného užívateľského rozhrania z jednotlivých menších celkov. Príklad použitia šablónovacieho systému Spacebars zo súboru `app/client/templates/report.html`:

```
{{#each tests}}
  <tr class="rowTest">
    <th scope="row">{{@index}}</th>
    <td class="testNameCol">{{name}}</td>
    <td>{{#if result}}PASS{{else}}FAIL{{/if}}</td>
  </tr>
  {{#each requirements}}
    <tr class="rowLine">
      <th scope="row">req {{@index}}</th>
      <td colspan="2">{{content}}</td>
    </tr>
  {{/each}}
{{/each}}
```

Dáta z databázy, ktoré chceme zobraziť v šablónach, sa definujú ako atribúty špeciálnej triedy `Template.`*nazov-sablony*`.helpers`, ktorá dáta sprístupní v jednotlivých šablónach. Celkové užívateľské rozhranie je rozdelené na 4 časti:

- navigácia,
- dokument,
- testovacie prípady,
- report,

pričom každému z nich odpovedá jedna šablóna v adresári `app/client/templates`. Všetky tieto šablóny sú zlúčené do jedného HTML súboru – `app/client/main.html`.

Meteor používa renderovaciu knižnicu *Blaze*[17], ktorá sa následne postará o „vykreslenie“ šablón do výslednej HTML stránky, ktorý je následne interpretovaný webovým prehliadačom.

4.3 Implementácia funkcionality

Pre názornejší popis implementácie funkcionality nástroja a zohľadnenie kľúčových aspektov sa správa projektov t.j. voľby ponuky tlačidla *Projects* popíšu až na konci tejto sekcie. Štruktúra tejto sekcie je nasledovný:

- popis spôsobu obsluhy užívateľských akcií obsahuje podsekcia 4.3.1,
- tvorba testovacích prípadov je opísaný v podsekcii 4.3.2,
- spôsob nahrania a zobrazenia špecifikačného dokumentu sa nachádza v podsekcii 4.3.3,
- vyznačovanie požiadaviek z dokumentu a ich priradenie testovacím prípadom je popísané v podsekcii 4.3.4,
- štruktúra ukladania dát na strane agenta je popísaný v podsekcii 4.3.5,
- popis tvorby testovacej šablóny sa nachádza v podsekcii 4.3.6,
- spusteniu testovacej sady sa venuje podsekcia 4.3.7,
- zobrazenie výsledkov testovania v klientskej časti nástroja je opísaný v podsekcii 4.3.8,
- popis tvorby reportu o aktuálnom pokrytí špecifikácie požiadaviek sa nachádza v podsekcii 4.3.9,
- správa projektov sa nachádza na konci kapitoly v podsekcii 4.3.10.

4.3.1 Obsluha udalostí

Obsluhy udalostí vo frameworku Meteor.js sa realizujú pomocou metódy triedy Template – `Template.`*nazov-sablony*`.events`, parametrom ktorého je objekt obsahujúci definície udalostí a im patriace obslužné funkcie. Uvedieme príklad:

```
Template.document.events({
  "click #browseBtn": function() {
    /* telo obsluznej funkcie */
  }
});
```

[17] Meteor Blaze a Spacebars – http://guide.meteor.com/blaze.html

4.3.2 Tvorba testovacích prípadov

Tlačidlo *Create* v dolnej časti tabuľky pre testovacie prípady slúži na vytvorenie testovacieho prípadu. Aplikácia detekuje udalosť kliknutia a vytvorí nový testovací prípad podľa schémy *Test-Case*. Atribúty naplní implicitnými hodnotami, pričom:

- farba testovacieho prípadu sa generuje náhodne pomocou nasledovného kódu:

```
var color = "\#000000".replace(/0/g, function() {
    return (~~(Math.random()*16)).toString(16);
});
```

  kde každá nula v reťazci sa nahradí pseudonáhodným hexadecimálnym číslom. Na generovanie pseudonáhodného čísla sa použila funkcia `Math.random()`[18],

- názov testovacieho prípadu je možné kedykoľvek zmeniť využitím knižnice *editable-text*, ktorá sa realizuje dvojklikom na názov testovacieho prípadu. Knižnica zaručí transformáciu textu na textový vstup. Zmena sa uloží stiskom tlačidla `Enter` alebo kliknutím mimo textového vstupu a textový vstup sa následne transformuje späť na text.

Každý testovací prípad v tabuľke obsahuje tlačidlo, ktoré vyvolá udalosť odstránenia vybraného testovacieho prípadu – odstráni sa z databázy aj spolu s požiadavkami, ktoré k nemu patria.

4.3.3 Nahranie a zobrazenie špecifikačného dokumentu

Kliknutím na tlačidlo *Browse* sa vykoná obslužná metóda, ktorá pomocou funkcie *FileReader*[19] klient asynchrónne prečíta obsah vybraného súboru z lokálneho FS užívateľa. Po získaní obsahu súboru sa vloží daný obsah do databázy, resp. priradí do dokumentu *document* kolekcie *Projects*.

Zobrazenie dokumentu sa realizuje pridaním nového atribútu do triedy `Template.document.helpers`, ktorej hodnota odpovedá samotnému dokumentu vloženému do databázy. O zobrazenie dokumentu sa už postará renderovací a šablónovací systém.

Užívateľ je pri nahrávaní dokumentu obmedzený maximálnou možnou veľkosťou BSON dokumentu Mongo kolekcie, ktorá je 16 MB[20]. Nahranie dokumentu je uskutočniteľný iba v prípade výberu textového súboru. Žiadne iné typy súborov v čase písania tejto práce nie sú podporované.

4.3.4 Vyznačovanie požiadaviek v dokumente a ich priradenie testovacím prípadom

Je dôležité si uvedomiť, že pri načítaní stránky webový prehliadač vytvorí DOM danej stránky. Každý DOM uzol má definovaný typ[21]. V našom dokumente sa môžu objaviť dva typy DOM uzlov – text alebo element (span element, ktorým sa už ohraničila časť textu). Pri snahe o ohraničenie vybraného textu nejakým elementom musíme myslieť na to, že to je možné iba v prípade, ak vybraný text sa nachádza v rámci jedného DOM uzlu.

[18] `Math.random()` – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random

[19] `FileReader()` – https://developer.mozilla.org/en-US/docs/Web/API/FileReader

[20] Limity MongoDB – https://docs.mongodb.org/manual/reference/limits/

[21] Typy DOM uzlov – https://developer.mozilla.org/en-US/docs/Web/API/Node

V prvom kroku je potreba zistiť, či vyznačený text je vôbec možné ohraničiť nejakým elementom. K vyznačenému textu pristupujeme prostredníctvom `window` objektu webového prehliadača, ktorý je súčasťou BOM[22]. `window.getSelection().getRangeAt(0)` vráti objekt typu `range`, ktorý reprezentuje rozsah vyznačeného textu v rámci DOM uzlu.

Po získaní „bezpečného“ rozsahu v dokumente, pristúpime k vytvoreniu *span* elementu, ktorým sa ohraničí daný rozsah. Vytvorenie realizuje funkcia `document.createElement()`, ktorej parametrom je názov elementu. Pomocu funkcie `setAttribute()` sa nastavia jej atribúty, konkrétne:

- *class = "docReq"*, ktorý sa ďalej využíva na odlíšenie vyznačeného textu v dokumente od nevyznačenej časti,
- *reqId = Random.id()*, kde `Random.id()` je funkcia frameworku Meteor, ktorá vygeneruje unikátny identifikátor a atribút span elementu *reqId*, ktorý jednoznačne identifikuje vyznačený text v dokumente,
- *style*, ktorý zaistí odlíšenie vyznačených textov v dokumente podľa príslušnosti k testovacím prípadom.

V momente, keď máme k dispozícii validný rozsah textu v dokumente a vytvorený span element, môžeme vykonať ohraničenie:

```
range.surroundContents(spanElement)[23];
```

Označená časť špecifikácie požiadaviek sa pridá do dokumentu `requirements` kolekcie *TestCases* a aktualizuje sa dokument *document* kolekcie *Projects* v databáze.

Odstránenie vyznačených častí textu v dokumente sa dosiahne kliknutím na vybranú časť textu. Nasledujúci kód odznačí vyznačený text z dokumentu:

```
$(event.target).contents().unwrap();
```

kde `event.target`[24] vráti element, ktorý aktivoval udalosť, resp. vyznačený text v dokumente, ktorý chce užívateľ odstrániť. Pomocou funkcie knižnice jQuery `contains()` získame obsah synovských uzlov vybraného elementu, v našom prípade jediného textového uzlu obsahujúci vyznačený text v dokumente, ktorý má užívateľ záujem odstrániť. Na konci aplikujeme na vybraný obsah ďalšiu funkciu knižnice jQuery `unwrap()`, ktorá odstráni jej otcovský element t.j. „vybalí“ ju z označenia. Na záver sa aktualizuje dokument rovnakým spôsobom ako pri vyznačení textu.

4.3.5 Štruktúra ukladania dát na strane agenta

V tejto podsekcii si stručne naznačíme, ako je implementovaný agent – akým spôsobom spracováva prichádzajúce AJAX požiadavky od klienta a popis systému ukladania informácií o projektoch a jeho testoch. Na uloženie týchto informácií je možno použiť buď databázu alebo využiť úložisko súborového systému. My sme sa rozhodli dáta ukladať na súborový systém kvôli jednoduchosti. Pri implementácii sa použili knižnice z `npm` – `shelljs`, `fs` a `path`.

Agent je postavený na Node.js, na ktorom je spustený framework Express.js. Ten obsahuje medzi inými aj modul HTTP, prostredníctvom ktorého máme prístup k routovacím

[22] **B**rowser **O**bject **M**odel – http://www.w3schools.com/js/js_window.asp

[23] `surroundContents()` – https://developer.mozilla.org/en-US/docs/Web/API/Range/surroundContents

[24] Event.target – https://developer.mozilla.org/en-US/docs/Web/API/Event/target

metódam `get()` alebo `post()`. Pomocou týchto metód sa implementujú jednotlivé prístupové body, ktoré sú definované v REST API a ich obslužné funkcie, ktoré vykonávajú požadované služby. Obslužné funkcie majú dva parametre – *request*, prostredníctvom ktorého máme prístup k parametrom požiadaviek a *response*, ktorý sa naplní dátami a je poslaný klientovi ako odpoveď na jeho požiadavku.

Na tvorbu testovacích šablón, spúšťanie testov a ukladanie projektov je potrebné, aby agent disponoval s nejakým systémom, na základe ktorého sa ukladajú informácie o projektoch a ich testoch – na strane agenta existuje adresár `projects`, v ktorom sa vytvárajú adresáre podľa názvov projektov (klient zaistí unikátnosť ich názvov). Do týchto adresárov sa ukladajú projekty (do súboru `project.json`) a vytvorené šablóny do súboru `tests/test.<ext>`. V čase písania tejto práce sa agent spolieha na túto štruktúru pri vykonávaní požiadaviek klienta.

4.3.6 Tvorba testovacej šablóny

V čase písania tejto práce nástroj podporuje generovanie testovacej šablóny iba v programovacom jazyku Python, pričom sa použil jeho vstavaný testovací framework `unittest`[25]. Požiadavkami na vytvorenie testovacej šablóny je existencia aspoň jedného testovacieho prípadu a unikátnosť názvov testovacích prípadov. Inak klientska časť nástroja nedovolí vyvolať udalosť kliknutia na tlačidlo *Template*.

Po úspešnom vyvolaní udalosti kliknutia klient zaistí validáciu názvov testovacích prípadov. Po validácii sa pristúpi k serializácii potrebných dát ku generovaniu testovacej šablóny:

- identifikátor projektu,
- názov projektu,
- programovací jazyk projektu,
- pole objektov testovacích prípadov, obsahujúce:
  - názov testovacieho prípadu,
  - pole vyznačených požiadaviek z dokumentu.

Dáta sa pošlú na prístupový bod `/template` pomocou metódy `POST`.

Agent zachytí požiadavku pomocou nasledovného kódu:

```
app.post("/template", function(request, response) {
  /* generovanie testovacej sablony */
});
```

kde parameter *request* callback funkcie obsahuje dáta poslané klientom. Nasleduje vetvenie kódu na základe programovacieho jazyka – priestor na jednoduchú podporu viacerých programovacích jazykov. Zavolá sa funkcia na vytvorenie Python unittest testovacej šablóny `pyUnittestTemplate(projName, tests)`, ktorá vráti vytvorenú šablónu testovacej sady podľa informácií obsiahnutých v jej dvoch parametroch – názov projektu sa použije pri vytvorení názvu testovacej triedy a tests sa použije na konštrukciu jednotlivých testovacích metód, v dokumentačnom reťazci ktorých sa uvedie zoznam príslušných vyznačených požiadaviek z dokumentu špecifikácii požiadaviek.

Musíme myslieť aj na potenciálny problém interpretácie znakov s interpunkciou alebo unikódových znakov, preto je dôležité vložiť do hlavičky testovacej šablóny reťazec:

[25] Unittest – https://docs.python.org/3.4/library/unittest.html

```
# −*− coding: utf−8 −*−
```

Zaistíme ním explicitne kódovanie zdrojového súboru v UTF-8[26]. Akonáhle sa šablóna vytvorila, môžeme pristúpiť k jej zapísaniu do súboru `projects/:projName/tests/test.py`, čo sa realizuje pomocou synchrónnej verzie funkcie `writeFile` z balíka `fs`. Jeden z parametrov tejto funkcie je aj kódovanie, kde sa taktiež nastavilo kódovanie UTF-8, čím sa zaručila správna interpretácia znakov. Podľa jej úspešnosti sa pošle späť odpoveď klientovi. Užívateľ je informovaný o úspešnosti vytvorenia šablóny prostredníctvom funkcie *Toastr*, ktorá zobrazí malý informačný panel v hornom pravom rohu užívateľského rozhrania. Klient ešte nastaví príznak `templated` (v kolekcii *Projects* v databáze) na pravdivú alebo nepravdivú hodnotu, v závislosti od úspešnosti vytvorenia testovacej šablóny a pridá časovú pečiatku do `templatedAt` dokumentu.

4.3.7 Spustenie testovacej sady

Podmienkou na začatie testovania je existencia testovacej sady na strane agenta. Klient sa riadi podľa príznaku `templated` v kolekcii *Projects*. Ak má hodnotu `true`, klient povolí vyvolanie udalosti kliknutia na tlačidlo *Execute*. Inak je toto tlačidlo zablokované.

Dôležité je vysvetliť, akým spôsobom je schopný užívateľ dopísať telá testovacích metód, ktoré sa vygenerovali v šablóne. Vzhľadom na skutočnosť, že implementovaný nástroj má spĺňať požiadavky, ktoré sú opísané v kapitole 3, kde sa nešpecifikuje spôsob dopísania testovacej šablóny a priloženie SUT[27], je na samotnom užívateľovi, ako sa vysporiada s touto situáciou. V čase písania tejto práce sa predpokladá, že nástroj bude spustený na počítači užívateľa, čiže bude mať prístup k súborovému systému, kam agent ukladá stavy projektov a generuje testovacie šablóny.

Klient pošle požiadavku na prístupový bod `/execute` pomocou metódy `GET`. Agent obdrží požiadavku rovnakým spôsobom, ako pri vytváraní testovacej šablóny až na použitú metódu `get()` namiesto `post()`. Parameter *request* callback funkcie obsahuje v tomto prípade iba názov projektu, podľa ktorého vyhľadá príslušný testovací súbor a podľa rozšírenia názvu súboru zistí, v akom jazyku bola testovacia sada vytvorená. Momentálne agent dokáže spúšťať iba testovacie sady programovacieho jazyka Python vstavaného testovacieho frameworku unittest.

Podľa názvu projektu sa poskladá príkaz spustenia testovacej sady:

```
testPath = "projects/" + projectName + "/tests";
runCmd = "python -m unittest discover -s " + testPath + " -p test.py";
```

kde:

- argumentom `-m unittest` sa spustí modul unittest,
- ktorému sa ako argument odovzdá voľba `discover`, ktorý dokáže vyhľadať testovací súbor v konkrétnom adresári,
- ktorý sa špecifikuje pomocou `-s testPath` a
- `-p test.py` udáva vzor, podľa ktorého nájde vyhovovujúci testovací súbor (predpokladá sa, že užívateľ nepremenuje daný súbor).

[26] Kódovanie UTF-8 – http://www.utf-8.sk/
[27] **S**ystem **U**nder **T**est – https://en.wikipedia.org/wiki/System_under_test

Uvedený príkaz sa odovzdá funkcii `exec` zo vstavanej knižnice Node.js `child_process`, ktorá vykoná daný príkaz. Funkcia `exec` sa vykoná asynchrónne, čize jej parametrom je callback funkcia, ktorá sa zavolá až po vykonaní daného príkazu. Obsahuje 3 parametre – návratový kód, štandardný výstup (stdout) a štandardný chybový výstup (stderr). Python unittest modul svoj výstup vypíše na stderr, ktorý slúži na získanie informácií o výsledku testovania.

Z výstupu sa získajú informácie, ktoré testovacie prípady vrátili `PASS` a ktoré vrátili `FAIL`. Pri neúspechu testovacieho prípadu sa získa i informácia, na ktorom tvrdení (angl. *assert*) daný test zlyhal a pod. Ako odpoveď sa pošlú serializované dáta na základe úspešnosti testovania. Klient následne obdrží odpoveď s výsledkami a notifikuje užívateľa pomocou knižnice *Toastr* o výsledkoch testovania. V nasledujúcej podsekcii sa popíše spôsob zobrazenia výsledkov v UI.

4.3.8 Zobrazenie výsledkov testovania

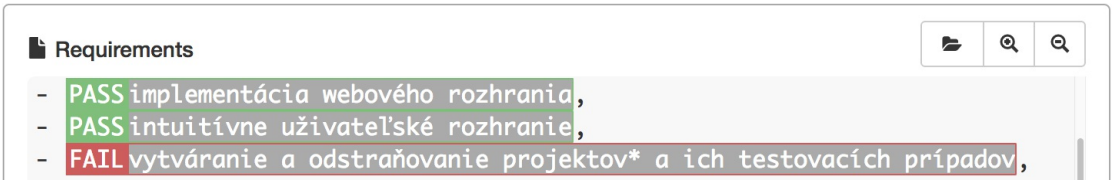
Výsledky sú k dispozícii, môže sa pristúpiť k zápisu výsledkov do databázy. Najprv sa vloží do kolekcie *Projects* informácia o úspešnosti spustenia testov (dokument `executed`), potom sa zapíšu výsledky testovania do dokumentu `result` kolekcie *Test-Cases*. V prípade, že pri testovaní došlo k nejakej chybe, tak sa nastaví i príznak `error` na hodnotu `true`, čo sa automaticky interpretuje v rámci nástroja, ako zlyhanie testovania.

V implementácii užívateľského rozhrania sa nachádzajú riadiace konštrukcie poskytované šablónovacím systémom Spacebars, ktoré umožňujú reflektovanie aktuálneho stavu testovania na klientskej časti (podľa jednotlivých príznakov v databáze). V tabuľke testovacích prípadov v UI sa nachádzajú dva stĺpce, ktoré informujú užívateľa o existencii testovacej šablóny na strane agenta a o výsledku posledného testovania. Zachytáva to obrázok 4.1. Na obrázku 4.2 je znázorený stav testovania v dokumente špecifikácie požiadaviek. Podľa príznaku `result` v kolekcii *Test-Cases* sa aplikujú príslušné CSS pravidlá na jednotlivé vyznačené časti textu v dokumente.

Obr. 4.1: Vzhľad zobrazenia informácií o existencii testovacej šablóny a o výsledku testovania v tabuľke testovacích prípadov. Na obrázku je zachytená situácia, keď sa testovacia šablóna úspešne vytvorila, ale testovanie sa skončilo neúspešne pre testovací prípad *TestCaseOne*.

4.3.9 Tvorba reportu

Tvorba reportu je výstupom nástroja. Obsahuje všetky dáta o danom projekte a poskytuje užívateľovi kompletný prehľad o aktuálnom pokrytí špecifikácie požiadaviek. K reportu vie užívateľ pristúpiť prostredníctvom tlačidla *Report*, ktoré sa nachádza na pravej časti UI. Po kliknutí na toto tlačidlo, server presmeruje užívateľa na stránku `/report`, kde zo-



Obr. 4.2: Vzhľad zobrazenia výsledku testovania v špecifikačnom dokumente. Na obrázku sú znázornené tri vyznačené požiadavky, z ktorých dve skončili úspešne (pred požiadavku sa pridalo označenie `PASS` a vytvoril sa zelený okraj) a jedna skončila neúspešne (označenie `FAIL` a červený okraj).

brazí stránku s reportom. Presmerovanie je realizované využitím knižnice `router` v súbore `app/lib/routes.js`.

Tvorba reportu je serializácia dát o danom projekte z databázy a správne zobrazenie pomocou renderovacieho a šablónovacieho systému. Na tvorbu reportu potrebuje aplikácia načítať z databázy nasledovné dáta:

- z kolekcie *Project* potrebuje názov projektu, špecifikačný dokument, programovací jazyk a časové pečiatky udalostí,
- z kolekcie *Test-Case* potrebuje vsetky testovacie prípady pre daný projekt a im priradené požiadavky zo špecifikačného dokumentu.

Príklad vytvoreného reportu sa nachádza v prílohe C.5.

V špecifikácii požiadaviek sa nachádza požiadavka *export reportu do HTML*. Túto činnosť nie je potrebné implementovať, keďže je vstavaná do dnešných webových prehliadačov – umožňujú stiahnuť stránku vo formáte HTML, ba dokonca umožňujú aj export do PDF.

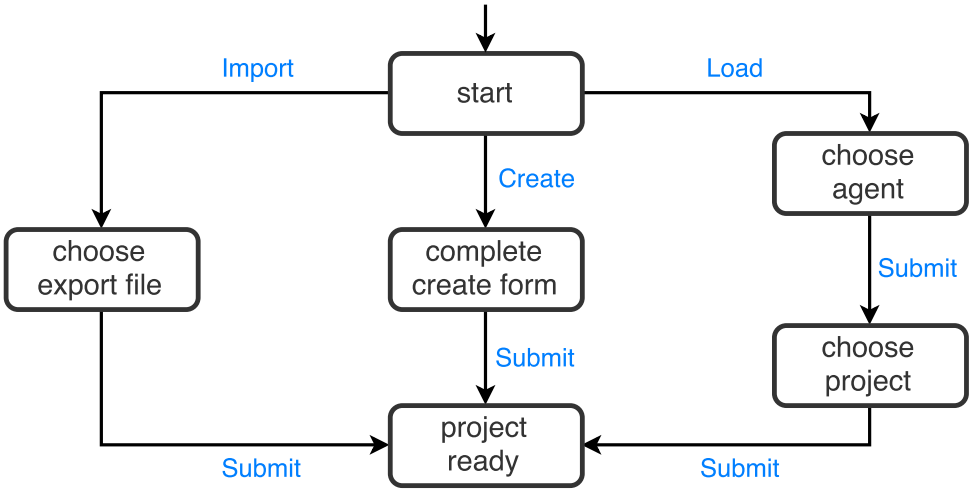
4.3.10 Správa projektov

Táto podsekcia popisuje implementáciu jednotlivých bodov špecifikácie požiadaviek 3.2, ktoré sa týkajú správy projektov:

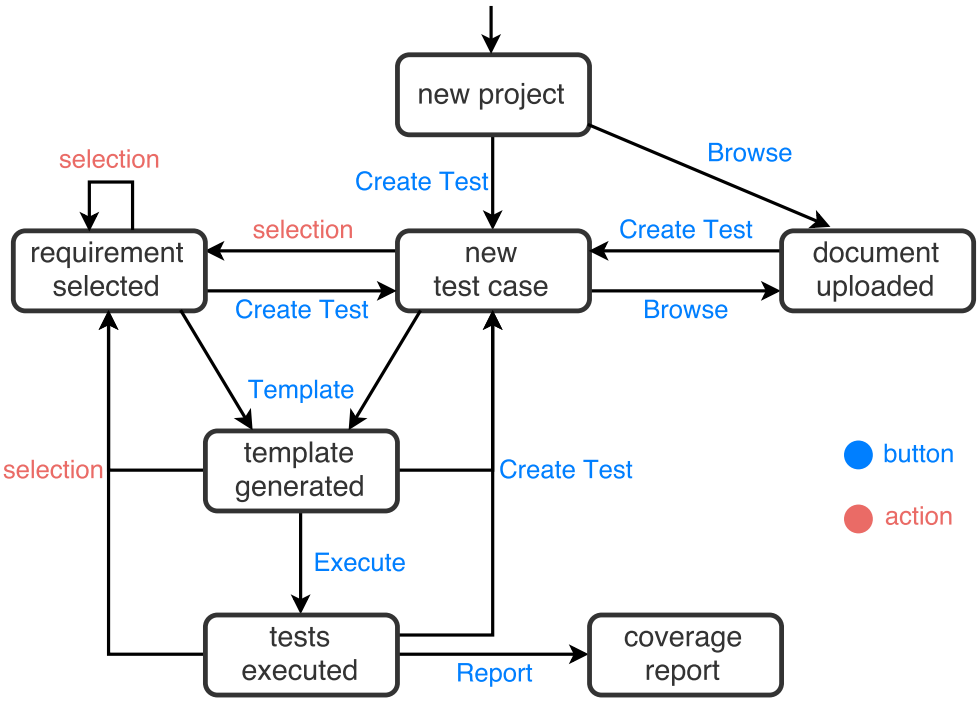
- Vytváranie a odstraňovanie projektov.
- Importovanie a exportovanie projektov.
- Uloženie a načítanie projektov.
- Prepínanie medzi existujúcimi projektami.

Všetky tieto činnosti sú spojené s jednotlivými možnosťami, ktoré sa nachádzajú v ponuke tlačidla *Project*. Kliknutím na niektorú z týchto možností sa rozozná patričná udalosť a zavolá sa obslužná funkcia.

Ako uvádza podsekcia 3.4.1, nástroj umožňuje vytvoriť projekt 3 spôsobmi. Tieto spôsoby sú zachytené na obrázku 4.3. Životný cyklus projektu nástroja sa nachádza na obrázku 4.4. Zachytáva stav novovytvoreného projektu a jednotlivé kroky, ktoré nástroj ponúka užívateľovi na vytvorenie reportu o aktuálnom pokrytí špecifikácií požiadaviek.



Obr. 4.3: Obrázok zachytáva možnosti vytvorenia projektu. Nástroj poskytuje 3 možnosti vytvorenia projektu – vytvoriť nový projekt, importovať projekt zo súborového systému a načítať uložený projekt na strane agenta.



Obr. 4.4: Obrázok zachytáva životný cyklus projektu v pomocnom nástroji pre testovanie na základe požiadaviek. Na obrázku je znázornený stav novovytvoreného projektu až po vytvorenie reportu o aktuálnom pokrytí špecifikácie požiadaviek.

Vytvorenie a odstránenie projektov

Vytvorenie nového projektu je reprezenované voľbou *Create*. Po kliknutí na túto voľbu sa objaví modálne okno, prostredníctvom ktorého užívateľ zadá názov projektu a vyberie

z ponuky programovací jazyk, v ktorom má záujem pracovať s daným projektom. Následne vyberie z výberu i URL adresu agenta, s ktorým chce komunikovať v rámci daného projektu. Po potvrdení zadaných údajov, klient vykoná validáciu vstupov – názov projektu musí byť unikátny, bez medzier a špeciálnych znakov. Je to dôležité z toho dôvodu, že názov projektu sa používa tak pri komunikácii s agentom, tak pri generovaní testovacej šablóny. Ak existuje v rámci nástroja aktívny projekt, tak sa nastaví na neaktívny a vloží sa novovytvorený projekt do databázy s nastavenými implicitnými hodnotami napr. príznak *templated*, čo nesie informáciu, či daný projekt má vytvorenú testovaciu šablónu sa nastaví na `false`. V poslednom kroku sa pošle AJAX požiadavka agentovi na prístupový bod `/create`, ktorý vytvorí v adresári `projects` adresár s názvom projektu a v ňom adresár `tests`.

Možnosť *Delete* zaisťuje odstránenie daného projektu. Server odstráni z databázy záznam o danom projekte a všetky záznamy o jeho testovacích prípadoch a vyznačených požiadavkách. Následne pošle AJAX požiadavku na prístupový bod agenta `/delete`, ktorý vymaže projekt z adresára `projects`.

Importovanie a exportovanie projektov

Možnosť exportovania projektu ponúka užívateľovi jednoduchú možnosť, ako uchovať stav rozpracovaného projektu stiahnutím na súborový systém. Na exportovanie aktuálneho stavu aplikácie je potrebné pristúpiť k potrebným dátam v databáze. Exportovanie sa uskutočňuje s využitím funkcie *Filesaver*, ktorá uloží serializovaný JSON objekt do súboru s názvom projektu a časovou pečiatkou do adresára *Downloads*.

Importovanie projektu zo súborového systému je inverzná udalosť voči exportovaniu. Pomocou funkcie *FileReader* aplikácia asynchrónne prečíta obsah vybraného súboru zo súborového systému užívateľa. Po získaní obsahu súboru s exportovanými dátami klient prevedie obsah do formátu JSON a vloží do databázy. Ak projekt už existuje v databáze, prepíše sa importovaným projektom. Inak sa vytvorí nový. Všetky zmeny sa propagujú aj agentovi pomocou požiadaviek AJAX – pri importovaní projektu sa posiela požiadavka na prístupový bod `/create`.

Uloženie a načítanie projektov

Uloženie sa uskutoční vybraním voľby *Save* v ponuke tlačidla *Project*. Podobne ako pri exportovaní, aj pri uložení sa najprv pripraví serializovaný objekt vo formáte JSON, ktorý obsahuje aktuálny projekt, všetky jeho testovacie prípady a priradené požiadavky z dokumentu. Pošle sa AJAX požiadavka na prístupový bod `/save` pomocou metódy *POST*. Agent informuje klienta o úspešnosti uloženia 3.3.3.

Načítanie projektu z agenta je operácia podobná exportovaniu projektu. Na rozdiel od exportovania serializované dáta o projektu získame od agenta. Po zvolení voľby *Load* v ponuke tlačidla *Project*, klient zobrazí modálne okno so zoznamom všetkých agentov. Užívateľ vyberie ten agent, z ktorého chce načítať projekt. Následne klient pošle požiadavku na prístupový bod `/savedProjects` vybraného agenta, ktorý vráti zoznam názvov všetkých uložených projektov – ide o názvy všetkých adresárov v adresári `projects`. Užívateľovi sa zobrazí modálne okno so zoznamom uložených projektov, kliknutím na ľubovoľný z nich sa pošle ďalšia požiadavka na prístupový bod agenta `/load` s potrebným parametrom – názvom projektu. Agent pristúpi k súboru `projects/:projName/project.json`, ktorý načíta do JSON objektu a pošle klientovi v odpovedi. Klient následne vloží získané dáta o vybranom projekte do databázy.

Prepínanie medzi existujúcimi projektami

Na konci ponuky tlačidla *Project* sa nachádza zoznam neaktívnych projektov. Kliknutím na vybraný neaktívny projekt užívateľ dosiahne transformáciu aktívneho projektu na neaktívny a zobrazenie vybraného projektu so všetkými jeho závislosťami. Je to dosiahnuté jednoduchou zmenou príznakov *active* v príslušných projektoch v databáze.

4.4 Známe obmedzenia

Prvým obmedzením aktuálnej verzie nástroja je potreba neustáleho „prekreslenia“ dokumentu počas vyznačovania častí textu dokumentu, čo sa prejaví posunom aktuálnej pozície v dokumente nahor na pôvodné miesto. Tento jav je nežiaduci hlavne v prípade, ak užívateľ pracuje s veľkým dokumentom a vykonáva vyznačenia textu v celom dokumente.

Druhé obmedzenie sa týka hraničných veľkostí dát. Základná konfigurácia balíka `body-Parser`, ktorý sa stará o spracovanie tela požiadavky `POST`, definuje maximálnu prípustnú veľkosť tela 1 MB[28]. Užívateľ môže naraziť na problém pri snahe o uložení projektu na strane agenta, ktorý obsahuje viac ako 1 MB dát. Veľkosť týchto dát je závislá od veľkosti textového dokumentu, ktorý užívateľ nahrá do aplikácie. Maximálna veľkosť BSON dokumentu v MongoDB databáze je 16 MB.

Ďalšie obmedzenie nástroja je možné spozorovať pri vytváraní testovacích prípadov. V čase písania tejto práce sa pri vytvorení testovacieho prípadu náhodne vygeneruje i farba 4.3.2, ktorá ho reprezentuje v rámci nástroja. Stáva sa, že sa vygeneruje rovnaký alebo veľmi podobný odtieň farby už existujúceho testovacieho prípadu. Ďalším problémom s náhodnými farbami je, že sa niekedy vygenerujú až príliš svetlé farby. To má za následok splynutie farby s bielym textom.

[28] BodyParser limity – https://github.com/expressjs/body-parser

Kapitola 5

Demonštrácia funkčnosti vytvoreného nástroja na netriviálnom prípade použitia

V tejto kapitole sa predvedie funkčnosť implementovaného nástroja. Jediným vstupom nástroja je špecifikácia požiadaviek. Demonštrácia sa uskutoční na jednej z požiadaviek nástroja – *importovanie projektu*. Vybraná požiadavka sa rozšírila o detaily tak, aby reflektovala reálne požiadavky na implementáciu importovania projektu v samotnom nástroji. Pri tvorbe špecifikácie sa použila technika recenzie nejednoznačnosti (ambiguity review), cieľom ktorej bolo upraviť alebo odstrániť časti špecifikácie, ktoré boli príliš vágne, nejasné, nejednoznačné alebo neúplné. Pri tejto činnosti sa postupovalo podľa [1], kde sú popísané slová alebo frázy, ktoré sú častými príčinami chýb v špecifikácii. Výsledné požiadavky sa nachádzajú v prílohe C.1.

Pri demonštrácii funkčnosti sa vytvorila séria obrázkov zobrazujúcich postup práce s vytvoreným nástrojom, ktoré sa nachádzajú v prílohe C. Cieľom tejto demonštrácie je ukázať postup práce s nástrojom, výstupom ktorého je report o aktuálnom pokrytí špecifikácie.

Na začiatku práce je potrebné vytvoriť nový projekt v nástroji. Uskutoční sa to kliknutím na tlačidlo *Project*, ktorý zobrazí ponuku s možnosťami navigácie. Z ponuky sa vyberie možnosť *Create*, čím sa objaví modálne okno, kde sa vyplnia potrebné údaje – názov projektu je *Project_Import*, keďže sa budú testovať požiadavky týkajúce sa importovania projektu. Nasleduje výber programovacieho jazyka (v čase písania tejto práce nástoj podporuje iba *Python Unittest*) a špecifikácia url adresy agenta. Tento stav je znázornený na obrázku C.1.

Po vytvorení projektu sa zobrazí názov projektu medzi tlačidlami *Project* a *Report*. Ďalším krokom je nahranie dokumentu do nástroja, ktorý je realizovaný kliknutím na tlačidlo *Browse* pod tlačidlom *Report*. Po kliknutí sa zobrazí modálne okno s výberom súboru, prostredníctvom ktorého sa vyberie súbor obsahujúci vytvorené požiadavky. Nasleduje proces tvorby testovacích prípadov, pri ktorom sa postupne vytvárajú testovacie prípady, vyznačia sa príslušné požiadavky v dokumente. Na obrázku C.2 sa nachádza aktuálny stav práce s nástrojom.

Nasleduje tvorba testovacej šablóny. Kliknutím na tlačidlo *Template* sa dosiahne vygenerovanie testovacej šablóny na strane agenta. Vygenerovaná šablóna pre uvedenú špecifikáciu a vytvorené testovacie prípady sa nachádzajú v prílohe C.2. Pre lepšiu demonštráciu sa po vygenerovaní šablóny spustilo testovanie pomocou tlačitka *Execute*. Výsledok je zachytený

na obrázku C.3. Keďže všetky uvedené tvrdenia v testovacej šablóne sú implicitne nútené vrátiť hodnotu `false`, výsledkom testovania je zlyhanie všetkých testovacích prípadov.

Vygenerovaná testovacia šablóna sa nachádza v súborovom systéme agenta. Po prístupu k šablóne sa doplnia telá jednotlivých testovacích metód, aby vrátili nejaké výsledky. Telá testovacích metód boli doplnené náhodným kódom, ktoré následne vrátili rôznorodé výsledky. Zobrazené výsledky testovania sú znázornené na obrázku C.4. Za účelom prehľadnejšieho zobrazenia aktuálneho pokrytia špecifikácie slúži výstupný report. Nachádza sa v prílohe na obrázku C.5 a na obrázku C.6.

Jednou z požiadaviek nástroja bola *spoľahlivosť funkcionality a konzistencia užívateľského rozhrania vo vybraných webových prehliadačoch – Google Chrome, Mozilla Firefox, Safari*. Pri demonštrácii sa použil webový prehliadač Chrome (verziu 50.0). Následne sa vykonala tá istá demonštráciá aj v ostatných 2 prehliadačoch (Mozilla Firefox verzia 45.0.1, Safari verzia 9.1) a môže sa prehlásiť, že nástroj funguje podľa špecifikácii.

Ďalšou požiadavkou nástroja bolo *intuitívne užívateľské rozhranie*. V priebehu práce bola venovaná väčšia priorita funkcionalite nástroja pred overovaním použiteľnosti. Z dôvodu absencie testovania použiteľnosti boli zvolené doporučené techniky návrhu užívateľského rozhrania.

Kapitola 6

Záver

Hlavným cieľom tejto práce bolo navrhnúť a implementovať pomocný nástroj pre testovanie na základe požiadaviek. Jediným vstupom nástroja je špecifikácia požiadaviek vo forme textového dokumentu. Nástroj umožňuje užívateľovi vytvoriť testovacie prípady a priradiť im požiadavky z dokumentu. Po priradení všetkých požiadaviek z dokumentu, nástroj dokáže vygenerovať testovaciu šablónu v jazyku Python C.2 prostredníctvom agenta, s ktorým komunikuje podľa navrhnutého REST API uvedené v prílohe B. Agent umožňuje užívateľovi aj vzdialené spúšťanie testovacej sady. Výsledky testovania pošle späť klientskej časti, ktorá ich následne zobrazí. Užívateľ má v každom prípade prehľad o aktuálnom pokrytí špecifikácie požiadaviek. Výstupom nástroja je report o aktuálnom pokrytí špecifikácie požiadaviek, ktorý je možné exportovať do formátu HTML alebo PDF podľa možností webového prehliadača.

6.1 Ďalší rozvoj nástroja

Implementovaný nástroj spĺňa požiadavky z kapitoly 3, avšak nasledujúce fázy vývoja nástroja by mali jednoznačne klásť veľký dôraz na testovanie a refaktorizáciu, ktorá uľahčí implementáciu ďalších rozšírení. Nasleduje popis možných rozšírení nástroja.

Podpora recenzie nejednoznačnosti (*ambiguity review*) vo forme sady funkcií, ktorá by umožnila kontrolu špecifikácie požiadaviek a upozornila uživateľa na potenciálne chyby v dokumente.

Podpora ďalších programovacích jazykov pri generovaní testovacích šablón. Toto rozšírenie zahrňuje aj spracovanie rôznych výstupov testovacích knižníc a vrátenie výsledkov testovania v jednotnom formáte.

Podpora grafu príčin a dôsledkov (*cause-effect graph*) vo forme interaktívneho nástroja na identifikáciu príčin a dôsledkov v špecifikácii, na tvorbu grafu príčin a dôsledkov a generovanie minimálnej testovacej sady, ktorá by pokrývala 100 % špecifikácie požiadaviek.

Podpora ďalších formátov špecifikačného dokumentu , ktoré sa v praxi bežne používajú. Jedná sa napr. o formáty PDF, DOC alebo XLS.

Literatúra

[1] Bender, R.: *The Ambiguity Review Process* [online]. http://benderrbt.com/Ambiguityprocess.pdf.

[2] Bender, R.: *Requirements Based Testing Process* [online]. http://benderrbt.com/Bender-Requirements%20Based%20Testing%20Process%20Overview.pdf, 2009.

[3] Beran, V.; aj.: ITU – *Webové technologie pro tvorbu uživatelských rozhraní.* Meteriály k prednáške. Interný dokument. [online]. https://www.fit.vutbr.cz/study/courses/ITU/private/lectures/Web/itu-webgui-cs.pdf.

[4] Berry, D. M.; aj.: *User's Manual as a Requirements Specification: Case Studies* [online]. Březen 2003.

[5] Castro, E.; Hyslop, B.: *HTML5 a CSS3 Názorný průvodce tvorbou WWW stránek.* Computer Press, 2012, ISBN ISBN 978-80-251-3733-8.

[6] Hruška, T.; aj.: IIS – *Architektury informačních systémů.* Metriály k prednáške. Interný dokument. [online]. https://www.fit.vutbr.cz/study/courses/WAP/private/opory/IIS100Architektury.pdf.

[7] Khan, E.: *Different Approaches to Black Box Testing Technique for Finding Errors* [online]. ročník 2, č. 4, Říjen 2011.

[8] Martin, J.: *An Information Systems Manifesto.* Prentice Hall, 1984, ISBN ISBN 978-0134647692.

[9] Mogyorodi, G. E.: *What Is Requirements-Based Testing?* [online]. *Crosstalk: The Journal of Defense Software Engineering*, Březen 2003.

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

[11] Skoković, P.; Rakić-Skoković, M.: *Requirements-based Testing Process in Practice* [online]. ročník 1, č. 4: s. 155–161.

[12] Smrčka, A.: ITS – *Testování na základe požadavků.* Materiály k prednáške. Interný dokument. [online]. https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ITS-IT/lectures/8-pozadavky.pdf?cid=10020.

[13] Zakas, N. C.: *Javascript pro webové vývojáře: programujeme profesionálně.* Computer Press, 2000, ISBN ISBN 978-80-251-2509-0.

[14] Zendulka, J.; aj.: IDS – *Úvod, základní pojmy databázových systémů.* Materiály k prednáške. Interný dokument. [online]. https://www.fit.vutbr.cz/study/courses/IDS/private/prednasky/1_uvod.pdf.

Prílohy

Zoznam príloh

Príloha A

Obsah DVD

```
/
├── afret/                  # koreňový adresár nástroja
│   ├── app/                # adresár webovej aplikácie
│   │   ├── .meteor/        # adresár s nastaveniami platformy Meteor.js
│   │   ├── client/         # adresár klientskej časti
│   │   │   ├── templates/  # HTML súbory (šablóny)
│   │   │   ├── stylesheets/ # CSS súbory
│   │   │   └── src/        # JS súbory
│   │   ├── server/         # adresár serverovej časti
│   │   └── lib/            # JS kódy routovania a databázových schém
│   └── agent/              # adresár agenta
│       ├── node_modules/   # adresár s nastaveniami platformy Node.js
│       ├── projects/       # koreňový adresár štruktúry projektov
│       └── src/            # zdrojové súbory agenta
├── doc/                    # zdrojové súbory textu tejto práce
└── bp.pdf                  # text práce v PDF
```

Príloha B

REST API

Vo všetkých prípadoch komunikácie sa používa serializačný formát JSON.

GET /create – vytvorí sa adresár s názvom daného projektu a v ňom sa vytvorí adresár `tests`. Adresár s názvom projektu slúži ako priestor na uloženie celého projektu, ktorý sa realizuje požiadavkou **POST /save**. V adresári `tests` sa vytvorí súbor, ktorý obsahuje testovaciu sadu pre daný projekt. Uskutoční sa to požiadavkou **POST /template**.

- **Parametre**
  - *projectName* - názov projektu.
- **Odpovede**
  - úspešné vytvorenie

```
{
  "projectId": "string",
  "created": true
}
```

  - neúspešné vytvorenie

```
{
  "projectId": "string",
  "created": false
}
```

GET /delete – odstráni sa adresár daného projektu a všetky jeho súbory a podadresáre.

- **Parametre**
  - *projectName* - názov projektu.
- **Odpovede**
  - úspešné odstránenie

```
{
  "deleted": true
}
```

  - neúspešné odstránenie

```
{
  "deleted": false
}
```

POST /save – uloženie aktuálneho stavu projektu a jeho testovacích prípadov do súboru `project.json` v adresári s názvom projektu.

- **Parametre**
  - *saveObject* =

```
{
  "project": {
    "id": "string",
    "name": "string",
    "language": "string",
    "document": "string",
    "agentUrl": "string",
    "templated": "boolean",
    "createdAt": "date",
    "savedAt": "date",
    "templatedAt": "date",
    "executedAt": "date",
    "active": "boolean"
  },
  "tests": [
    {
      "id": "string",
      "projectId": "string",
      "name": "string",
      "requirements": [
        {
          "reqId": "string",
          "content": "string"
        }
      ],
      "color": "string",
      "executed": "boolean",
      "result": "boolean",
      "error": "boolean"
    }
  ]
}
```

- **Odpovede**
  - úspešné uloženie

```
{
  "projectId": "string",
  "saved": true
}
```

  - neúspešné uloženie

```
{
  "projectId": "string",
```

```
  "saved": false
}
```

GET /savedProjects – vráti pole názvov všetkých uložených projektov na strane agenta.

- **Parametre**
  - *void*
- **Odpovede**
  - názvy uložených projektov

```
{
  "savedProjects": [ "string" ]
}
```

GET /load – vráti *saveObject*, ktorý je načítaný zo súboru `project.json` nachádzajúci sa v adresári s názvom projektu.

- **Parametre**
  - *projectName* - názov projektu.
- **Odpovede**
  - saveObject (viď **POST /save** parameter)

POST /template – vytvorí sa šablóna testovacej sady v programovacom jazyku, ktorý si užívateľ vybral pri vytvorení daného projektu. Šablóna sa uloží do adresára `tests` ako `test.(ext)`.

- **Parametre**
  - *projectId* - identifikátor projektu v databázy na server časti nástroja,
  - *projectName* - názov projektu,
  - *projectLanguage* - názov vybraného programovacieho jazyka pri vytvorení projektu,
  - *tests* - pole objektov testovacích prípadov pre daný projekt.
- **Odpovede**
  - úspešné vytvorenie šablóny

```
{
  "projectId": "string",
  "templated": true
}
```

  - neúspešné vytvorenie šablóny

```
{
  "projectId": "string",
  "templated": false
}
```

GET /execute – spustí sa testovacia sada, ktorá bola vytvorená službou **POST /template**. Predpokladá sa, že pred jej spustením užívateľ dopíše telo testovacích prípadov vo vytvorenej šablóne testovacej sady.

- **Parametre**
  - *projectName* - názov projektu na serveri.
- **Odpovede**
  - testovacia sada nebola nájdená

```
{
  "projectName": "string",
  "executed": false,
  "error": true,
  "errorMessage": "string"
}
```

  - všetky testy boli úspešné

```
{
  "projectName": "string",
  "status": "PASS",
  "totalTests": "integer"
}
```

  - aspoň jeden test nebol úspešný

```
{
  "projectName": "string",
  "status": "FAIL",
  "totalTests": "integer",
  "totalErrors": "integer",
  "failedTests": "integer"
}
```

  - chyba pri spustení testov

```
{
  "projectName": "string",
  "status": "ERROR",
  "errorMessage": "string"
}
```

Príloha C

Testovanie

C.1 Špecifikácia požiadaviek

Importovanie projektu

V navigačnej ponuke v klientskej časti nástroja existuje tlačidlo reprezentujúce importovanie projektu. Na toto tlačidlo je naviazaná udalosť kliknutia. Kliknutím na tlačidlo importovania sa zavolá obslužná funkcia. Obslužná funkcia spĺňa nasledujúce požiadavky:

- zabezpečí zobrazenie dialógového okna na výber súboru zo súborového systému.
- Výber je umožnený iba pre súbory formátu `application/json`.
- Po zvolení súboru sa načíta jeho obsah do JSON objektu, v prípade neúspechu sa vypíše chybové hlásenie o chybe pri načítaní obsahu a importovanie sa skončí neúspešne. Inak sa pokračuje ďalej.
- Ak existuje v databáze projekt, ktorého dokument `active` sa rovná hodnote `true`, táto hodnota sa zmení na hodnotu `false`, inak sa pokračuje ďalej.
- Načítaný JSON objekt sa zvaliduje podľa schém v databáze. Ak schémam nevyhovuje, vypíše sa chybové hlásenie o "nesprávnom formáte načítaného objektuä importovanie sa skončí neúspešne. Inak sa pokračuje ďalej.
- Pošle požiadavku agentovi na vytvorenie projektu pomocou technológie AJAX, v ktorej je špecifikované:
  - URL adresa agenta s prístupovým bodom `/create`,
  - názov načítaného projektu,
  - časový limit požiadavky.

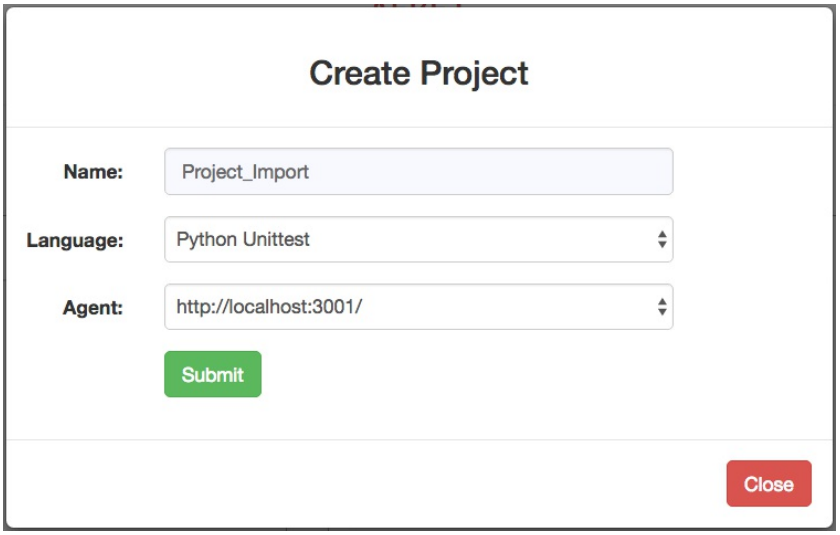
  Ak je požiadavka úspešná, vloží sa exportovaný projekt do databázy na serveri. Vypíše sa hlásenie o úspechu importovania a importovanie sa skončí úspešne. Inak sa vypíše chybové hlásenie o neúspešnom poslaní AJAX požiadavky na importovanie a importovanie sa skončí neúspešne.

C.2 Testovacia šablóna

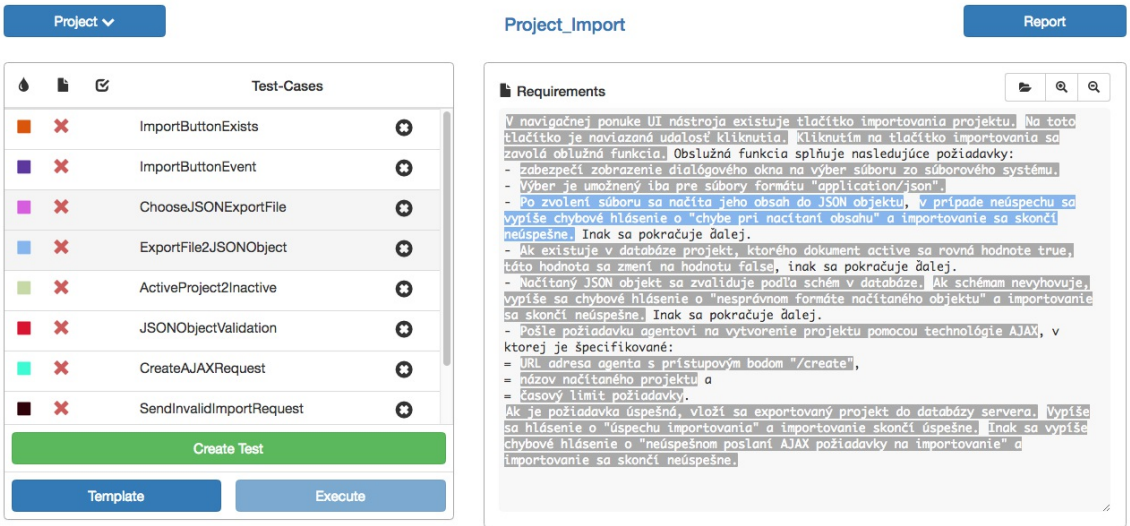
```
# !/usr/bin/env python
# -*- coding: utf-8 -*-

import unittest

class TestProject_Import (unittest.TestCase):

    def test_ImportButtonExists (self):
        """Selected lines in document:
        1 'V navigačnej ponuke UI nástroja existuje tlačítko
           importovania projektu.'
        """
        # write your code here
        self.assertTrue(False)

    def test_ImportButtonEvent (self):
        """Selected lines in document:
        1 'Na toto tlačítko je naviazaná udalosť kliknutia.'
        2 'Kliknutím na tlačítko importovania sa zavolá obslužná funkcia
           .'
        """
        # write your code here
        self.assertTrue(False)

    def test_ChooseJSONExportFile (self):
        """Selected lines in document:
        1 'zabezpečí zobrazenie dialógového okna na výber súboru zo sú
           borového systému.'
        2 'Výber je umožnený iba pre súbory formátu "application/json".'
        """
        # write your code here
        self.assertTrue(False)

    def test_ExportFile2JSONObject (self):
        """Selected lines in document:
        1 'Po zvolení súboru sa načíta jeho obsah do JSON objektu'
        2 'v prípade neúspechu sa vypíše chybové hlásenie o~"chybe pri
           načítaní obsahu" a importovanie sa skončí neúspešne.'
        """
        # write your code here
        self.assertTrue(False)

    def test_ActiveProject2Inactive (self):
        """Selected lines in document:
        1 'Ak existuje v~databáze projekt, ktorého dokument active sa
           rovná hodnote true, táto hodnota sa zmení na hodnotu false'
```

```
        """
        # write your code here
        self.assertTrue(False)

    def test_JSONObjectValidation (self):
        """Selected lines in document:
        1 'Načítaný JSON objekt sa zvaliduje podľa schém v~databáze.'
        2 'Ak schémam nevyhovuje, vypíše sa chybové hlásenie o~"nesprá
           vnom formáte načítaného objektu" a importovanie sa skončí ne
           úspešne.'
        """
        # write your code here
        self.assertTrue(False)

    def test_CreateAJAXRequest (self):
        """Selected lines in document:
        1 'Pošle požiadavku agentovi na vytvorenie projektu pomocou
           technológie AJAX, v~ktorej je špecifikované:'
        2 'Pošle požiadavku agentovi na vytvorenie projektu pomocou
           technológie AJAX'
        3 'URL adresa agenta s~prístupovým bodom "/create"'
        4 'názov načítaného projektu'
        5 'časový limit požiadavky'
        """
        # write your code here
        self.assertTrue(False)

    def test_SendInvalidImportRequest (self):
        """Selected lines in document:
        1 'Inak sa vypíše chybové hlásenie o~"neúspešnom poslaní AJAX po
           žiadavky na importovanie"'
        2 'importovanie sa skončí neúspešne.'
        """
        # write your code here
        self.assertTrue(False)

    def test_SendValidImportRequest (self):
        """Selected lines in document:
        1 'Ak je požiadavka úspešná, vloží sa exportovaný projekt do
           databázy servera.'
        2 'Vypíše sa hlásenie o~"úspechu importovania"'
        3 'importovanie skončí úspešne'
        """
        # write your code here
        self.assertTrue(False)

if __name__ == '__main__':
    unittest.main()
```

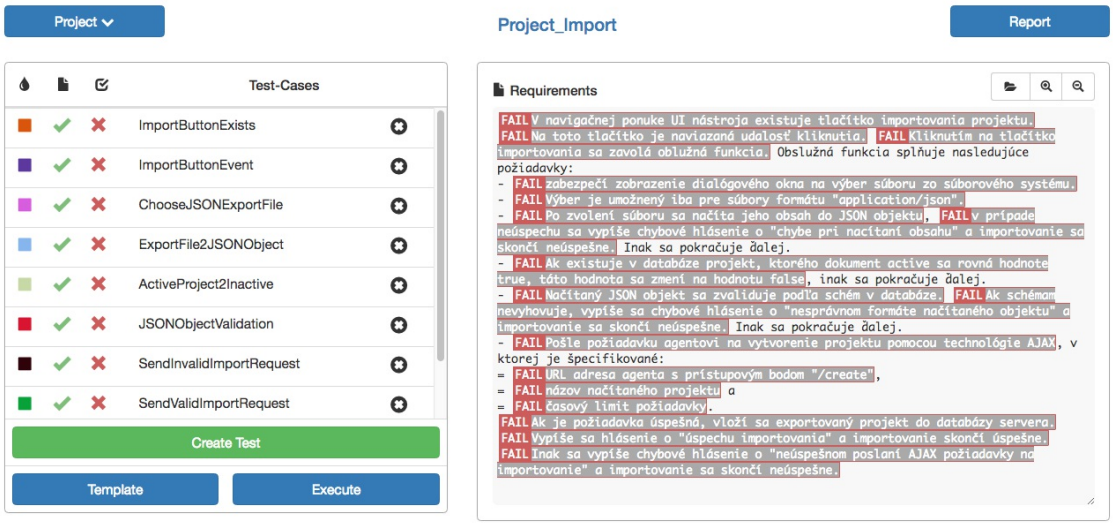
C.3 Obrázky zachytávajúce prácu s nástrojom



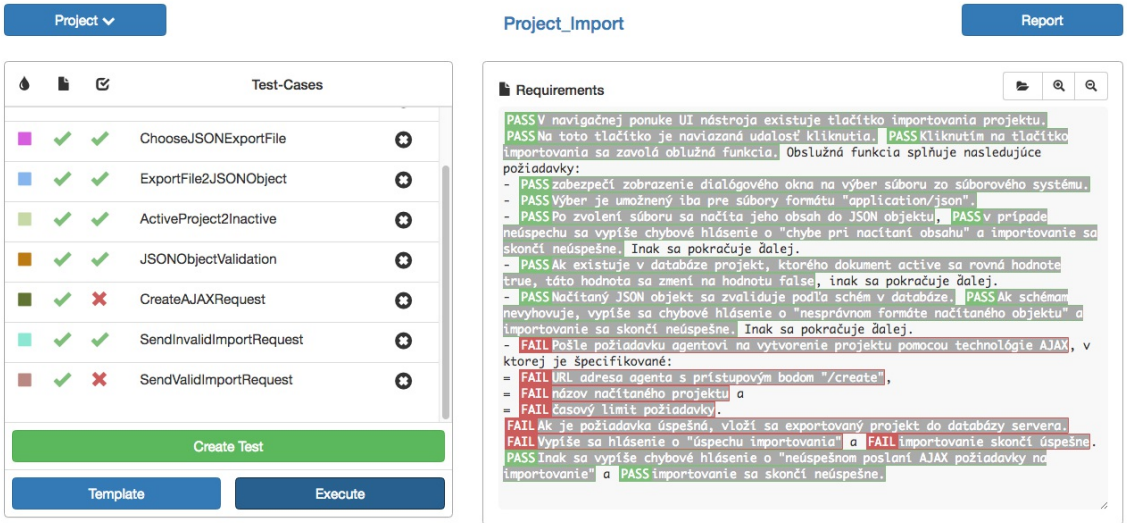
Obr. C.1: Obrázok zachytávajúci modálne okno, prostredníctvom ktorého užívateľ zadáva vstupné parametre na vytvorenie nového projektu. Je potrebné, aby zadal názov nového projektu, vybral programovací jazyk a URL adresu agenta.



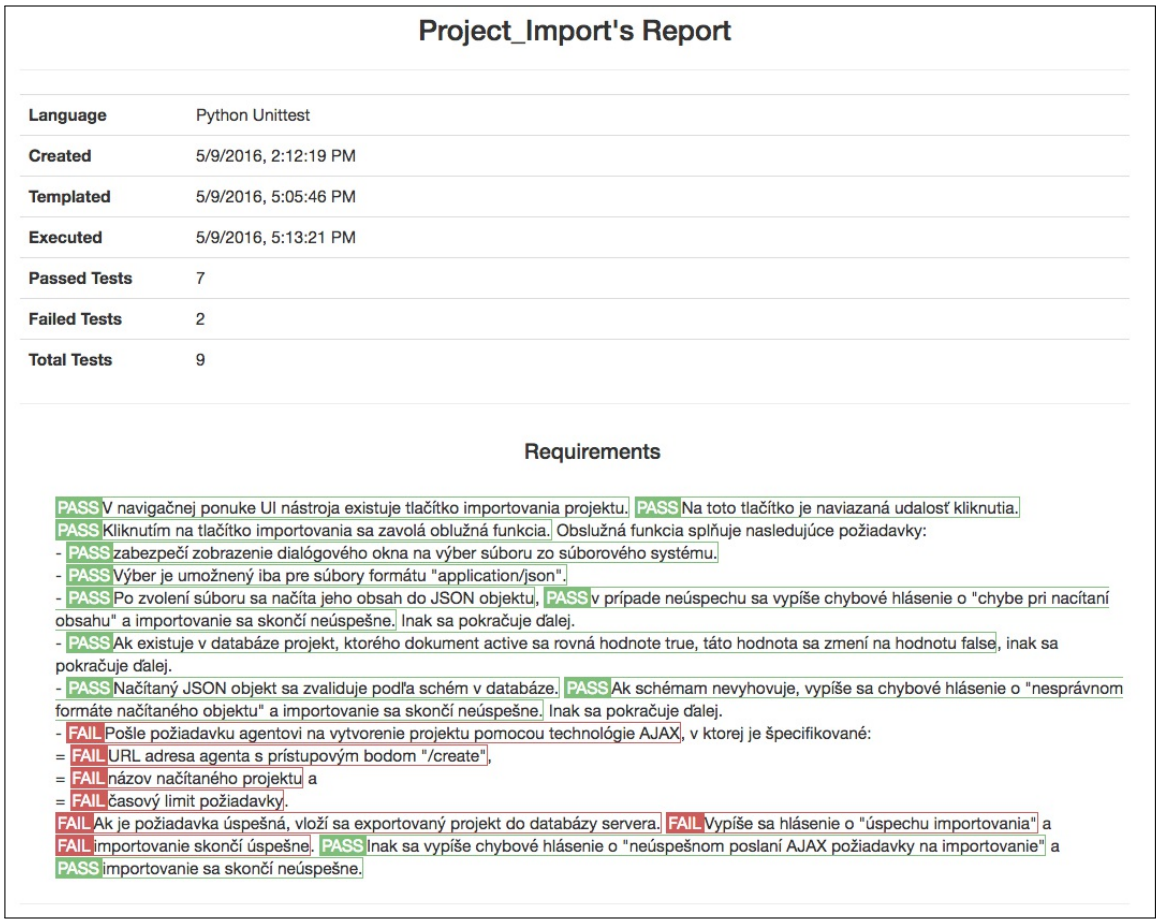
Obr. C.2: Na obrázku je zachytený stav, kedy sú vytvorené testovacie prípady a sú k nim priradené jednotlivé požiadavky z dokumentu RS. Druhý stĺpec v tabuľke testovacích prípadov obsahuje červené krížiky, čo znamená, že testovacia šablóna ešte nebola vytvorená.



Obr. C.3: Obrázok zachytáva stav nástroja po vytvorení testovacej šablóny a spustení testov. Druhý stĺpec v tabuľke testovacích prípadov poukazuje na úspešné vytvorenie testovacej šablóny a tretí stĺpec predstavuje výsledok testovania. Na pravej strane obrázku môžeme pozorovať výraznú zmenu dokumentu spôsobenú zobrazením výsledkov testovania pre každú označenú požiadavku.



Obr. C.4: Na obrázku môžeme vidieť zmenu v zobrazení výsledkov testovania po modifikácii niektorých testovacích prípadov. V našom prípade všetky testovacie prípady okrem *CreateAJAXRequest* a *SendValidImportRequest* vráti PASS, čo sa prejavilo aj v treťom stĺpci tabuľky, aj v dokumente.



Project_Import's Report

Language	Python Unittest
Created	5/9/2016, 2:12:19 PM
Templated	5/9/2016, 5:05:46 PM
Executed	5/9/2016, 5:13:21 PM
Passed Tests	7
Failed Tests	2
Total Tests	9

Requirements

PASS V navigačnej ponuke UI nástroja existuje tlačítko importovania projektu. PASS Na toto tlačítko je naviazaná udalosť kliknutia.
PASS Kliknutím na tlačítko importovania sa zavolá oblužná funkcia. Obslužná funkcia spĺňuje nasledujúce požiadavky:
- PASS zabezpečí zobrazenie dialógového okna na výber súboru zo súborového systému.
- PASS Výber je umožnený iba pre súbory formátu "application/json".
- PASS Po zvolení súboru sa načíta jeho obsah do JSON objektu, PASS v prípade neúspechu sa vypíše chybové hlásenie o "chybe pri nacítaní obsahu" a importovanie sa skončí neúspešne. Inak sa pokračuje ďalej.
- PASS Ak existuje v databáze projekt, ktorého dokument active sa rovná hodnote true, táto hodnota sa zmení na hodnotu false, inak sa pokračuje ďalej.
- PASS Načítaný JSON objekt sa zvaliduje podľa schém v databáze. PASS Ak schémam nevyhovuje, vypíše sa chybové hlásenie o "nesprávnom formáte načítaného objektu" a importovanie sa skončí neúspešne. Inak sa pokračuje ďalej.
- FAIL Pošle požiadavku agentovi na vytvorenie projektu pomocou technológie AJAX, v ktorej je špecifikované:
= FAIL URL adresa agenta s prístupovým bodom "/create",
= FAIL názov načítaného projektu a
= FAIL časový limit požiadavky.
FAIL Ak je požiadavka úspešná, vloží sa exportovaný projekt do databázy servera. FAIL Vypíše sa hlásenie o "úspechu importovania" a FAIL importovanie skončí úspešne. PASS Inak sa vypíše chybové hlásenie o "neúspešnom poslaní AJAX požiadavky na importovanie" a PASS importovanie sa skončí neúspešne.

Obr. C.5: Na obrázku sa nachádza výsledný prehľadový report. Na hornej časti sú uvedené rôzne informácie o projekte *Project Import*. Na dolnej časti je zobrazený dokument RS. Jednotlivé požiadavky sú označené na základe výsledku testovania pre lepší prehľad o aktuálnom pokrytí.

5	**ActiveProject2Inactive**	PASS
-	Ak existuje v databáze projekt, ktorého dokument active sa rovná hodnote true, táto hodnota sa zmení na hodnotu false	
6	**JSONObjectValidation**	PASS
-	Načítaný JSON objekt sa zvaliduje podľa schém v databáze.	
-	Ak schémam nevyhovuje, vypíše sa chybové hlásenie o "nesprávnom formáte načítaného objektu" a importovanie sa skončí neúspešne.	
7	**SendInvalidImportRequest**	PASS
-	Inak sa vypíše chybové hlásenie o "neúspešnom poslaní AJAX požiadavky na importovanie"	
-	importovanie sa skončí neúspešne.	
8	**SendValidImportRequest**	FAIL
-	Ak je požiadavka úspešná, vloží sa exportovaný projekt do databázy servera.	
-	Vypíše sa hlásenie o "úspechu importovania"	
-	importovanie skončí úspešne	

Obr. C.6: Obrázok zachytáva časť druhej časti prehľadového reportu, ktorá obsahuje zoznam jednotlivých testovacích prípadov a im patriace požiadavky.