ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Webová služba na testování REST API
Student:	Michal Kváček
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Cílem práce je vytvořit webovou aplikaci, na které si programátor bude moci testovat REST API. Programátor si v přehledném grafickém rozhraní definuje parametry API (metody, vstupní parametry). Na základě takto definovaných parametrů zvolí sadu dotazů, vstupních dat a definuje k nim očekávané výstupy. Takto vytvořené testy bude aplikace spouštět a testovat tím tak zvolené API.

Proveďte průzkum existujících konkurenčních řešení.
Navrhněte webovou aplikaci včetně vhodného grafického prostředí s ohledem na maximální jednoduchost a použitelnost.
Na základě návrhu realizujte funkční prototyp webové aplikace.
Proveďte vhodné testy použitelnosti a funkčnosti prototypu.
Na základě výsledků testů realizujte finální aplikaci.
Aplikaci realizujte jako open-source.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 7. prosince 2015

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

Bakalářská práce

Webová služba na testování REST API

Michal Kváček

Vedoucí práce: Ing. Jiří Hunka

16. května 2016

Poděkování

V první řadě bych rád poděkoval své přítelkyni Michaele, která mi byla oporou a trpělivě snášela mé ne vždy příjemné nálady. Dále bych chtěl poděkovat svým rodičům, kteří mě zahrnuli veškerou myslitelnou morální podporou, svému kolegovi a kamarádovi Peterovi za jeho cenné rady a připomínky a v neposlední řadě samozřejmě všem přátelům a známým, které jsem využil pro potřeby uživatelského testování.

Prohlášení

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

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

V Praze dne 16. května 2016 .....

Odkaz na tuto práci

Kváček, Michal. *Webová služba na testování REST API* . Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Předložená bakalářská práce se zabývá analýzou a následnou implementací systému pro testování REST API, který bude realizován jako webová služba. Výsledná aplikace je naprogramována jako jednostránková aplikace za použití javascriptového frameworku AngularJS. Mimo samotnou implementaci aplikace obsahuje tato práce i analýzu existujících řešení a otestování uživatelského prostředí pomocí Nielsenovy heuristické analýzy.

Klíčová slova REST API, testování, spolupráce v týmu, open source

Abstract

This bachelor thesis consists of technical analysis followed by implementation of web application used for testing REST API. This application is built on top of javascript framework AngularJS as one-side application. Except implementation, this thesis contains analysis of existing services and constructing Nielson's heuristic analysis for testing of user interface.

Keywords REST API, testing, team cooperation, open-source

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Vytvoření větší aplikace se v dnešní době neobejde bez testování, které zaručují, že aplikace funguje dle očekávání. Díky zavedení systémového testování lze odhalit mnoho chyb již rané fázi vývoje, tudíž je pochopitelné, že řádné testování aplikace už v rámci jejího programování dokáže nejen zrychlit proces vývoje, ale zároveň ho i zefektivnit a přispět k celkové udržovatelnosti projektu. Proto jsem se rozhodl vypracovat v rámci bakalářské práce aplikaci, která zjednoduší proces testování aplikací postavených na REST API.

V následujícím textu nejdříve uvedu existující služby, následně provedu analýzu potřeb a očekávání uživatelů. Na tento průzkum bych rád navázal technickou analýzou završenou implementací funkční aplikace. V práci se chci zaměřit na návrh uživatelského prostředí, který se samozřejmě neobejde bez otestování přístupnosti.

Cílem práce není vytvoření aplikace, která by funkční výbavou mohla konkurovat již existujícím a vyspělým službám. Cílem je nabídnout alternativu v oblasti testování menších SPA aplikací.

Dílčím (a čistě mým osobním) cílem je seznámení se s novými technologiemi, které budou vybrány pro účely implementace.

Kapitola 1

Testování REST API

1.1 Co je to REST API?

REST je zkratka pro *Representational State Transfer*. Jedná se o způsob návrhu architektury aplikace. První idea tohoto typu architektury byla uvedena roku 2000 Royem Fieldingem [1].

REST popisuje přístup k datům, kde každý zdroj[1] je definován jako URI[2]. Nad každým zdrojem REST definuje metody přístupu - získání (`GET`), vytvoření (`POST`), editace (`PUT`) a smazání (`DELETE`).

REST sám o sobě není vázán na žádný konkrétní protokol, nicméně vzhledem k jeho velkému využívání při návrhu webových a mobilních aplikací[1] je provozován hlavně na protokolu HTTP [3].

1.2 Možnosti testování

Pro jednoduchost a využití v rámci mé bakalářské práce budu v následujícím textu uvažovat REST postavený na HTTP protokolu.

Pro přístup k jednotlivým zdrojům, resp. jejich URI je zapotřebí nástroj, který umožňuje odeslat požadavky metodou `GET`, `POST`, `PUT` a `DELETE`. (Občas se používá ještě `OPTION` pro zjištění, jaké HTTP metody jsou podporovány serverem.)

Jeden z nástrojů je `curl`. Jedná se o nástroj dostupný z příkazového řádku, umožňující nastavit širokou škálu parametrů odeslaných jako HTTP požadavek [2]. Pro účely testování se `curl` samo o sobě nepoužívá, neboť vyžaduje poměrně velké úsilí na kontrolu odpovědi serveru. Nicméně `curl`, resp. knihovna `libcurl` je implementována v mnoha programovacích jazycích, díky čemuž

[1] Zdrojem mohou být data nebo třeba stav aplikace
[2] Uniform Resource Identifier
[3] Hypertext Transfer Protocol

vytváří více či méně robustní řešení se zachováním funkcionality samotného `curl`. Příkladem může být PHP [3] nebo Python [4].

Uživatelsky přívětivějším způsobem testování API jsou nástroje vytvářející dotazy pomocí `curl` (resp. `libcurl`) za uživatele. Nabízí grafické prostředí, kde si uživatel může do textových polí vložit vstupní data. Tyto nástroje nejsou složité na naprogramování, některé frameworky (pro PHP např. Restler) nabízejí generování tohoto prostředí automaticky [5]. Tato metoda je příjemná pro vývoj, resp. procházení API, avšak pro testování je nepohodlná. Tento způsob - stejně jako předchozí - vyžadují kontrolu od uživatele.

Dalším stupněm je ruční psaní testů pomocí mnoha nástrojů (Jasmine, behat, ...). Výhoda těchto technologií je jejich variabilita. Použití těchto nástrojů s sebou nese i nutnost naučit se zapisovat testy v požadovaném formátu (jazyce).

Posledním krokem, který zmíním, je kombinace posledních dvou řešení - tedy grafická aplikace umožňující spouštět testy a zároveň kontrolovat jejich výsledek, což je tématem mé bakalářské práce.

Kapitola 2

Existující služby

Vzhledem k faktu, že tvorba API není v dnešní době nic neobvyklého, existuje několik nástrojů, které řeší stejný (nebo podobný) problém jako moje bakalářská práce. Při průzkumu existujících řešení jsem našel služby Apiary, Runscope a vREST. Z desktopových aplikací se krátce zmíním i o SoapUI.

Jednotlivé služby zde krátce představím a zhodnotím jejich funkční výbavu a následně i uživatelské prostředí pomocí Nielsenovy heuristické analýzy.

2.1 Hodnocení funkcionality

V této části hodnocení se zaměřím na funkcionalitu bez přihlédnutí k uživatelskému prostředí. Následující text bude obsahovat popis funkcionality, případně slovní doplnění.V poslední části textu budou poznatky shrnuty do stručné tabulky.

Dostupnost a podpora

- Dostupnost na různých platformách (prohlížečích, mobilních zařízení)
- Dostupnost a přehlednost nápovědy
- Lokalizace aplikace
- Cena - pro jednotlivce, tým do 10 osob a větší tým

Funkcionalita

- Možnosti typů validací (definice vlastního validátoru, seznam dostupných validátorů atp.)
- Řetězení dotazů (předávání parametrů z odpovědi jednoho dotazu do požadavku druhého dotazu)

- Podpora datasetů pro spouštění testů
- Organizace testů do logických celků (vytváření tzv. *test-case*)

Podpora práce v týmu

- Podpora projektů
- Podpora týmových rolí (správa oprávnění pro jednotlivé role/členy)

Integrace do aplikací třetích stran

- Integrace do nástroje pro kontinuální integraci (API, plugin, ...)
- Integrace do project management nástrojů (Jira, Redmine, Track, ...)

2.2 Nielsenova heuristická analýza

Nielsenova heuristická analýza je způsob, jakým lze posoudit přístupnost a použitelnost uživatelského rozhraní. Analýza spočívá v porovnání aktuálního stavu aplikace se seznamem heuristik, jejichž seznam je uveden na adrese `https://www.nngroup.com/articles/ten-usability-heuristics/`.

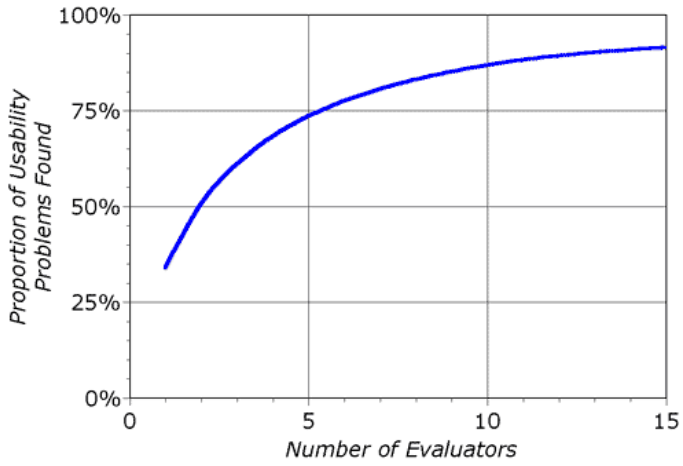
Tento způsob hodnocení je většinou prováděn alespoň třemi experty, neboť se očekává, že každý odborník dokáže odhalit pouze určitou část možných problémů. Podle Jacoba Nielsena platí, že každý hodnotitel odhalí zhruba 35 % problémů [6]. Zároveň ale upozorňuje na fakt, že se zvyšováním počtu hodnotitelů neroste počet nalezených problémů lineárně. Tento vztah ilustruje obrázek 2.1.

2.2.1 Postup při provádění analýzy

Pro účely této práce budu Nielsenovu heuristickou analýzu provádět pouze sám. Účelem této práce není detailně rozebírat existující nástroje. Nicméně pro objektivní hodnocení všechny nalezené nedostatky v jedné aplikaci prověřím i ve všech zbývajících. Věřím, že se mi tímto přístupem podaří heuristickou analýzu provést důkladně a v míře dostatečné pro účely této práce.

Vzhledem k faktu, že heuristiky jsou definována možná až velmi obecně, dovolím si heuristickou analýzu provést podle heuristik uvedených v předmětu BI-TUR v roce 2016. Tato pravidla byla zjednodušena a upravena pro přímé použití v oblasti webových aplikací [7]. Konkrétně se jedná o tyto návodné otázky (převzato z [7]):

1. Ví uživatel vždy, kde se nachází (navigace). Ví uživatel vždycky, v jakém je aplikace stavu?



Obrázek 2.1: Vztah počtu hodnotitelů na počet nalezených problémů v přístupnosti. Převzato z [6].

2. Jsou názvy, popisky a jinak používané termíny v souladu s terminologií cílové skupiny?

3. Jsou názvy, popisky a jinak používané termíny srozumitelné cílové skupině? Jsou názvy kategorií (například v menu) voleny s ohledem na srozumitelnost pro cílovou skupinu?

4. Poskytuje každá akce zpětnou vazbu srozumitelnou pro uživatele?

5. Může se uživatel dostat z každé situace? Obejde se takové opuštění bez nutnosti opakovat dlouhou sekvenci akcí? Je použití tlačítek $\leftarrow$ a $\rightarrow$ správné?

6. Jsou všechny objekty, akce a jiné prvky vidět, kdykoliv je jich potřeba, aniž by si je uživatel musel pamatovat?

7. Jsou všechny akce, uspořádání a význam konzistentní s očekáváním cílové skupiny (zvyky odjinud, ...)?

8. Odpovídá vzhled aplikace a ovládacích prvků cílové skupině?

9. Je plocha zobrazení využita přiměřeně (vzhledem k celkovému vzhledu)?

Po vyhodnocení uvedených heuristik sestavím nalezené problémy do stručné tabulky, kde budou jednotlivé problémy ohodnoceny i závažností. Stupně závažností jsem zvolil tyto: *triviální*, *malá*, *střední*, *vysoká* a *znemožňující používání*. Na závěr celé analýzy budou tyto výsledky shrnuty do souhrnné tabulky.

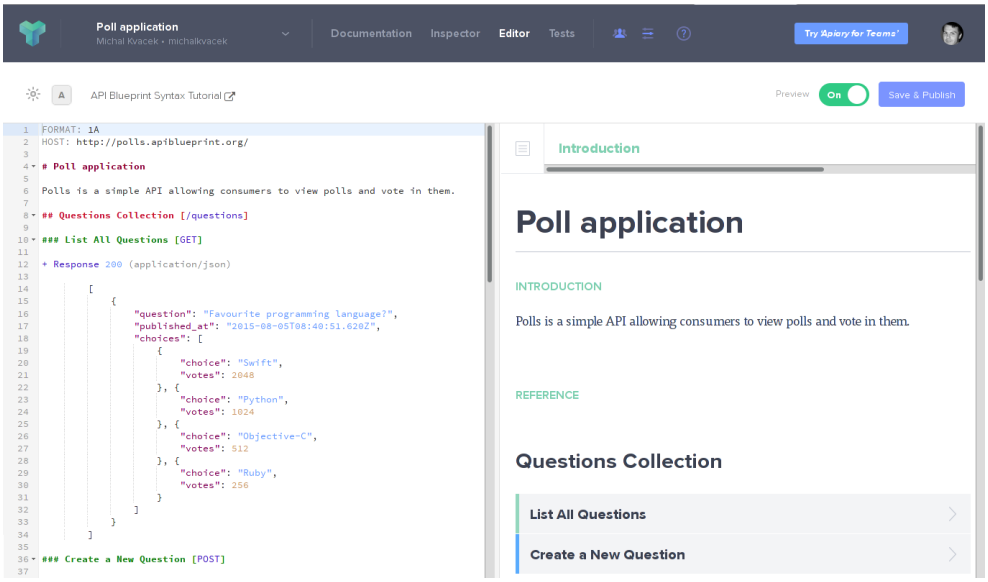
2.3 Apiary

Obrázek 2.2: Apiary: logo

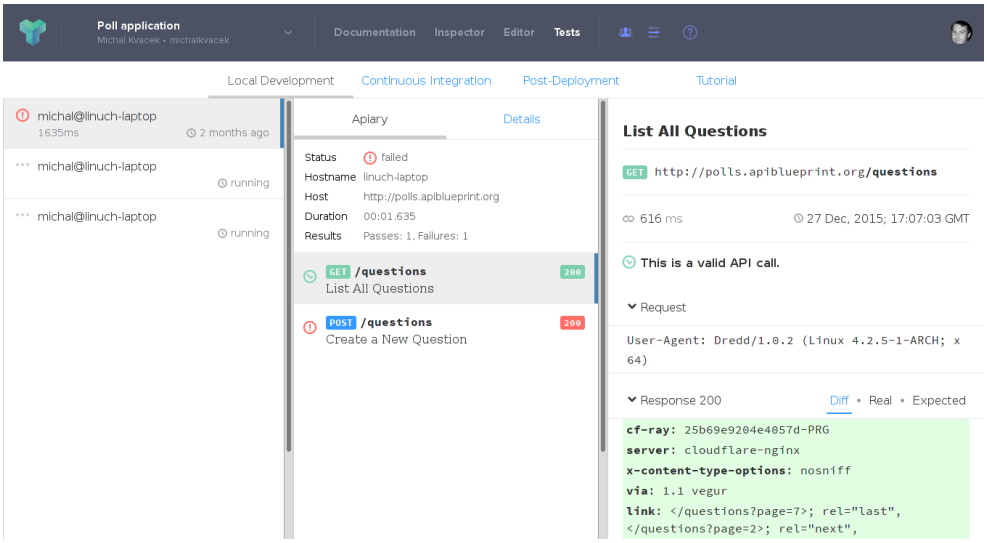
- URL: `https://apiary.io`

- Dokumentace: `https://help.apiary.io/`

Apiary je velmi zajímavý nástroj na udržování kvality API serverů. Jeho primární zaměření je na tvorbu dokumentace, nicméně nabízí i možnosti testování pomocí aplikace *dredd*.



Obrázek 2.3: Apiary: editor s živým náhledem



Obrázek 2.4: Apiary: přehled proběhnutých a běžících testů

Cílové skupiny uživatelů

Apiary je nástroj zaměřený na tvorbu dokumentace k API, který jako druhotnou službu nabízí jednak mockování serverů a jednak jejich testování, identifikoval jsem tři cílové skupiny.

Jednou z nich jsou *vývojáři* či *testeři API*, kteří sami doplňují dokumentaci, případně využívají již existující API Blueprint pro ověření funkcionality implementovaných metod. U těchto uživatelů lze předpokládat předchozí zkušenosti jak s tvorbou dokumentace, tak i s IT terminologií a psaním kódu.

Druhou skupinou uživatelů budou přímo *tvůrci dokumentace*. Tato skupina uživatelů nemusí mít s psaním kódu přímé zkušenosti, nicméně lze předpokládat základní znalosti v oblasti IT a s tím spojenou terminologii.

Poslední identifikovaná skupina – skupina konzumentů API a jeho dokumentace – pro účely této práce nejsou příliš zajímaví, neboť jejich činnost se omezuje fakticky jen na čtení vygenerované dokumentace. K samotnému procesu testování nemají vůbec přístup a proto nejsou pro účely této heuristiky důležití.

Hodnocení funkcionality

Apiary není primárně určeno pro testování API. Jeho účel je tvorba dokumentace. Z tohoto důvodu dopadlo nejhůře v hodnocených nástrojích pro testování API serverů, to však neznamená, že se jedná o špatnou či nepoužitelnou službu!

Pro účel testování se používá NodeJS aplikace `dredd` společně s popisem API pomocí Blueprintu. V prvním kroku je potřeba inicializovat samotný

`dredd`. Tato inicializace si vyžádá cestu k Blueprintu a další potřebné věci. Toto nastavení se samozřejmě dělá jen jednou. Samotné testování je potom zajištěno spuštěním příkazu dredd. Pro lepší názornost přikládám výstup z konzole.

```
[michal: ~] $ dredd init -r apiary -j \
apiaryApiKey:22b282d9...0782e2e0...0b15e4a \
-j apiaryApiName:michalkvacek
? Location of the API blueprint /home/michal/apiary.apib
? Command to start API backend server e.g.
(bundle exec rails server)
? URL of tested API endpoint http://polls.apiblueprint.org/
? Programming language of hooks nodejs
? Dredd is best served with Continuous Integration.
Create CircleCI config for Dredd? No

Configuration saved to dredd.yml

Run test now, with:

$ dredd

[michal: ~] $ dredd
Configuration dredd.yml found, ignoring other arguments.
info: Using apiary reporter.
info: Beginning Dredd testing...
pass: GET /questions duration: 933ms
pass: POST /questions duration: 467ms
complete: 2 passing, 0 failing, 0 errors, 0 skipped, 2 total
complete: Tests took 2258ms
complete: See results in Apiary at:
https://app.apiary.io/...a6d4889bb77a
```

Apiary je dostupné pro malé týmy (do 5 uživatelů) zdarma, ovšem bez možnosti spravovat přístupová oprávnění jednotlivých uživatelů. Pro větší týmy (do 20 členů, případně pro povolení správy oprávnění) je služba zpoplatněna 99 $ za měsíc [8]. Na vyzkoušení je však k dispozici 30 dní zkušební doby, což je dostatečně dlouhá doba pro otestování a seznámení se se službou.

Co se testování API týče, Apiary (resp. dredd) využívá dokumentaci samotného API. Díky tomu se testuje vždy to, co je zdokumentováno. Na druhou stranu ale chybí pokročilé možnosti, které nabízí konkurenční služby - např. v podobě definice vlastního validátoru, extrakce proměnných z odpovědi, stejně jako výběr dat pro testování z nějakého datasetu.

Funkcionalitu pro testování API pro nástroje kontinuální integrace řeší Apiary přes zmiňovaný `dredd`. Tento nástroj nabízí při inicializaci své konfigurace i vytvoření konfiguračního souboru pro nástroj pro kontinuální integraci CircleCI (`https://circleci.com`). Možnosti jiné integrace jsem nenašel.

Heuristická analýza

Rozhraní této služby je velmi jednoduché a subjektivně příjemné. Uživatel ví vždy, kde se nachází a to hlavně proto, že v levé části horního menu je vidět aktuálně editovaný projekt a ve vedlejším menu je zvýrazněna zobrazená stránka. Celkový koncept uživatelského prostředí dle mého názoru odpovídá očekávání cílových skupin. Každá akce provedená v systému poskytuje viditelnou zpětnou vazbu, orientace v prostředí je po prvních pár minutách práce poměrně intuitivní. Funkce tlačítek prohlížece pro pohyb v historii funguje přesně tak, jak uživatel očekává a při pohybu v aplikaci jsem nezaznamenal jedinou akci, ze které bych se nemohl dostat žádným jednoduchým způsobem. Samotné prostředí uživateli nedává ani moc prostoru pro chybu, pokud pomineme nevalidní zápis Blueprintu. Uživatelsky velmi příjemnou vlastností je „živý náhled“ editované dokumentace. Celkový koncept uživatelského prostředí, rozmístění prvků a pohyb v aplikaci odpovídá zvykům cílových skupin.

Popisky použité v aplikaci jsou srozumitelné minimálně pro první dvě uvedené cílové skupiny, tedy vývojáře/testery API a tvůrce dokumentace. Jediný drobný problém, který jsem v tomto směru objevil, je tlačítko „Inspector“. Zde by uživatel pravděpodobně očekával něco jiného než přehled dotazů směřující na mockované API metody. Nicméně pojmenování akcí a prvků je konzistentní.

Nápověda je v systému viditelná, je umístěna na očekávaném místě, tedy v pravé části horní navigace. Mimo odkaz na dokumentaci nabízí přímo i odkaz na seznam otázek na známém fóru StackOverflow.

Co se tedy uživatelského prostředí týče, je navrženo přehledně, je konzistentní a neobjevil jsem žádné problémy v použitelnosti.

3. Srozumitelnost terminologie	
Inspektor jako odkaz na mockované metody	nízká

Tabulka 2.1: Odhalené problémy při používání Apiary

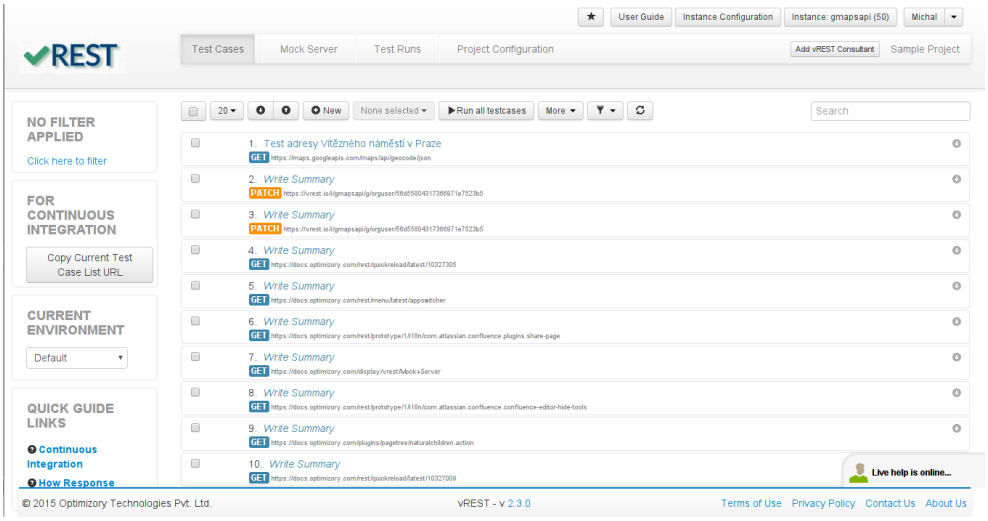
2.4 vREST

- URL: `https://vrest.io`
- Dokumentace: `https://docs.optimizory.com/display/vrest`

Služba vREST je webová služba, která si klade za cíl zjednodušit testování REST API služby.

Obrázek 2.5: vREST: logo

Kromě samotného testování nabízí i rozhraní na mockování API serverů. Přestože vREST funguje jako webová aplikace, pro spouštění testů je potřeba prohlížeč Google Chrome. Samotný prohlížeč však nestačí, je zapotřebí doinstalovat ještě rozšíření [9].



Obrázek 2.6: vREST: titulní strana projektu

Cílové skupiny uživatelů

Oproti Apiary se vREST již zaměřuje na testování API serverů, proto cílovou skupinou uživatelů budou především *vývojáři* a *testeři* API. Pro testera bude důležité, aby mohl jednoduše nahlásit nalezený problém do nějakého issue trackeru, pro vývojáře bude nezbytné umět rychle identifikovat, kde vznikl problém. Znalosti terminologie, technologií, postupů a věcí spojených s vývojem lze očekávat velmi srovnatelné. Z toho důvodu je pro účely heuristiky budu považovat za jedinou cílovou skupinu.

Hodnocení funkcionality

Klíčovou vlastností je samozřejmě testování API. V tomto směru vREST poskytuje očekávané funkce, tedy vytvoření *testcase* (requestu) a přiřazení *assertion* (vyhodnocovacích pravidel). Další možností validace odpovědi ze serveru

je kontrola oproti *schématu* JSON dokumentu, a to formátem označovaným jako *draft-03*, případně *draft-04*. Třetí možností kontroly je pak vlastní *validátor*. Jedná se o kód v Javascriptu, který lze použít pro komplexnější kontroly.

Samotné testy lze organizovat ještě do jednotlivých prostředí. Relativně snadno lze tak rozlišit mezi požadavky vedoucími na produkční API a požadavky vedoucími na testovací verzi.

V administraci se nachází i přehled všech spuštěných testů, odkud je možné vzniklé problémy nahlásit do issue trackeru Jira [10]. Možnost integrace do jiného systému jsem neobjevil. Mimo uvedené integrace je k dispozici API pro nástroje kontinuální integrace. Dle oficiální dokumentace [11] je připraven modul pro nástroj Jenkins [12], případně lze použít nástroj přímo od vRESTu - *vrunner*. Použitím druhé možnosti by mělo být možné integrovat testy do libovolného integračního nástroje [13].

Jedná se o placenou službu. Cena je dána za uživatele, ne za celý tým. Každý uživatel stojí 10 $ měsíčně [14]. Tato možnost nabízí určitě vyšší volnost než konkurenční cenové nabídky.

V rámci prémiových služeb je možné do projektu přidat i tzv. *vREST konzultanta*, který bude mít přístup jen pro čtení (pokud není nastaveno jinak) a může týmu pomoci s nastavením a používáním služby. Další částí podpory je živý chat. Operátor sice není připojen 24 hodin denně, ale cca od 8. hodiny ranní do 16. hodiny odpolední hlásí status „online“.

Jak jsem uvedl v úvodním textu o této službě, je vyžadován doplněk do prohlížeče Google Chrome. Pomocí tohoto doplňku lze jednoduše „nahrát“ vykonávané requesty. Nástroj sice jen sbírá požadavky odesílané z prohlížeče, které následně ukládá do databáze vRESTu, ale i tak se jedná o celkem příjemné usnadnění práce.

Heuristická analýza

Uživatelské prostředí je děleno standardně na horní menu, levý sidebar a obsahovou část. V horní části je zobrazena navigace, kde uživatel jasně vidí aktuálně zobrazenou stránku. Po prvním přihlášení jsem chvíli musel hledat tlačítko, případně nějaký jiný ovládací prvek, kterým bych mohl vytvořit nový test. Celkově je plocha zobrazení využita vhodně, nicméně na menším rozlišení jsem narazil na problém, že pravý sidebar se nevejde na výšku okna a vzhledem k jeho ukotvení na stránce není možné zakrytý obsah zobrazit.

Zpětná vazba je jasně viditelná a srozumitelná. V případě, že aplikace neodpoví ihned, zobrazí se nepřehlédnutelná „načítací“ animace. V případě spuštění samotných (v terminologii vREST) *testcase* je zobrazen animovaný *progress bar*, na kterém je jasně zřetelné, kolik testcase proběhlo a kolik jich ještě zbývá. Zároveň barevně odlišuje úspěšné a neúspěšné volání.

Drobnou nepříjemností je, že některé formuláře nelze potvrdit prostým stisknutím klávesy Enter, ale je vyžadováno kliknutí myší. Vzhledem k faktu,

že služba je stavěna pro uživatele zvyklé používat hlavně klávesnici (vývojáře), jedná se o nepříjemnou věc.

Terminologie se zdá téměř v souladu se zvyky cílových skupin, nicméně jsem narazil na následující problémy. vREST označuje jako `testcase` jeden `request`, ke kterému může uživatel přiřadit validační pravidla – *assertions*. Dále zavádí ještě pojem `testsuite`, což je skupina jednotlivých `testcase`. A pro spuštěné skupiny `testcase` zavádí pojem `test`. Myslím, že tato pojmenování by mohla mezi uživateli působit zmatek. Mimo pojmů souvisejících přímo s testováním, zavádí vREST ještě pojem `instance` pro označení (pravděpodobně) projektu. O konkrétním významu tohoto pojmu si nejsem jistý. V přehledu jednotlivých *testcase* v konkrétní *instanci* se v horní navigaci zobrazuje záložka *Project configuration*. A *Project settings* je rozdílné nastavení, než *Instance settings*. Nechť už pojmy znamenají cokoliv, napříč celou aplikací jsou konzistentní, takže po překonání prvotního zmatku se vývojář/tester v systému zorientuje.

Většina prvků je dobře viditelných, na problém jsem narazil pouze v nastavení, když jsem se snažil přidat nového uživatele. V horní navigaci jsem klikl na *Project configuration*, dále pak na *Users*, kde jsem očekával možnost zaslání pozvánky, případně jinou akci vedoucí k přiřazení uživatele. Tato možnost se zde ale nevyskytuje – postup, jakým lze přiřadit uživatele k projektu, jsem nalezl až v dokumentaci. Před samotným přidáním uživatele ze záložky *Project configuration* je potřeba ho vytvořit v *Instance settings*. Následně se zde zobrazí v seznamu a lze ho do projektu přiřadit. Tento postup rozhodně není přívětivý a bez přečtení dokumentace (na kterou bohužel z *Project configuration* odkaz nevede) i těžce realizovatelný.

Dostupnost a přehlednost nápovědy je (stejně jako u Apiary) dobrá. Mimo ikonku v menu však vREST zobrazuje přímé odkazy na nápovědu u velkého množství akcí, které uživatel může se systémem provádět. Samotná dokumentace je pak opatřena i screenshoty z aplikace. Procházení historie funguje dle očekávání.

Věřím, že po překonání nesrovnalostí s terminologií, seznámením se s konceptem *instance* a *projektu* a osvojením si určitých principů v používání systému je vREST spolehlivým nástrojem pro testování API. Nicméně tyto překážky určitě dokáží odradit nejednoho uživatele.

2.5 Runscope

- URL: `https://www.runscope.com`
- Dokumentace: `https://www.runscope.com/docs`

Jako třetí službu uvedu Runscope. Oproti Apiary, Runscope je nástroj zaměřený čistě na testování a monitoring API. Umí sledovat jak uptime [4],

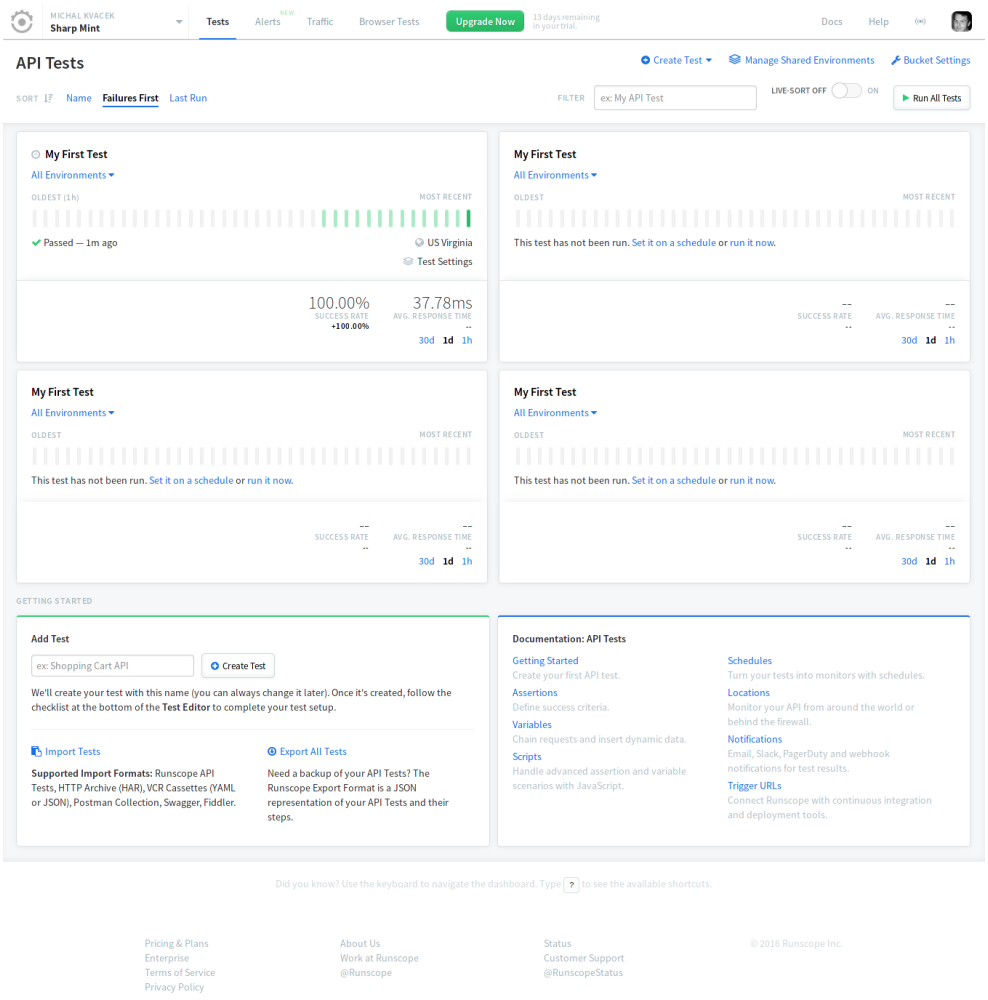
[4] Dostupnost serveru

3. Srozumitelnost terminologie	
Test, testsuite, testcase	vysoká
Nejasný význam pojmu *instance*	vysoká
Nastavení projektu a instance	střední
6. Viditelnost prvků	
Tlačítko pro přidání testu	nízká
Přiřazení uživatele	vysoká
Skrytí části sidebaru při menším rozlišení	střední
7. Očekávaná funkčnost ovládacích prvků	
Nemožnost odeslat formulář klávesou Enter	střední

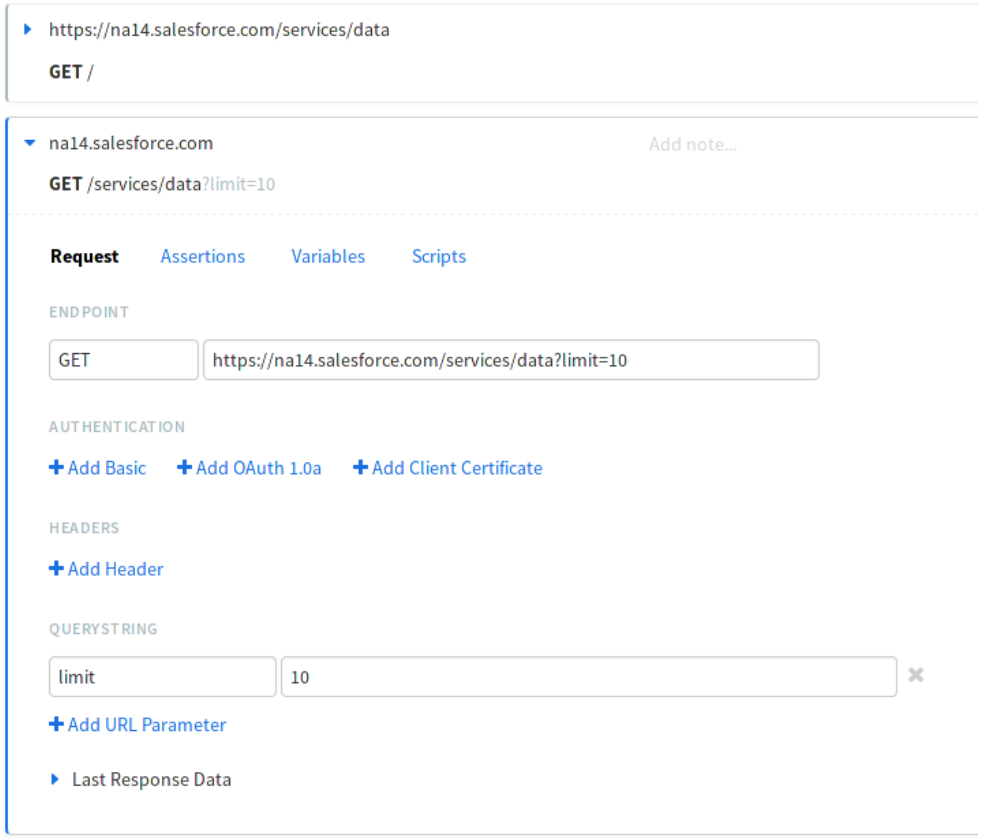
Tabulka 2.2: Odhalené problémy při používání vREST

Obrázek 2.7: Runscope: logo

tak i dobu odezvy na server. Z takto získaných dat vytváří grafy a reporty uživateli.



Obrázek 2.8: Runscope: dashboard



Obrázek 2.9: Runscope: detaily dotazu

Hodnocení funkcionality

Dle funkční výbavy, variability nastavení, možností výběru lokality pro spouštění testů a mnohých dalších vlastností lze soudit, že Runscope je na trhu buď nejdéle, nebo do jeho vývoje bylo investováno nejvíce prostředků. Tato vlastnost se však odráží i na měsíčních poplatcích, které nejsou zrovna nízké (detaily viz tabulka 2.4).

Samotné testování probíhá podobně jako u služby vREST. Každý *request* patří do nějakého *testu*, který je součástí projektu (v terminologii používané v Runscope *bucketu*). U každého dotazu lze definovat pravidla, případně schéma odpovědi nebo vlastní *script* (pravidlo vyjádřené jako Javascriptová funkce). Tato funkcionalita se tedy příliš neliší od dříve zmiňované služby vREST. Bonusovou funkcí, kterou však vREST nemá, je pak periodické spouštění testů, nastavení geografické lokality, odkud se bude provádět požadavek, možnost definice skriptu spuštěného před vlastním testem a notifikace na základě výsledku testu.

U definice testu (resp. přiřazování) *requestů* má uživatel možnost nastavení

časové prodlevy mezi requesty, jejich pořadí a dokonce i větvení pomocí definovaných podmínek. To znamená, že Runscope umožňuje vložení „skupiny *requestů*“, které se vykonají jen tehdy, pokud konkrétní vybraná proměnná splňuje zvolené kritérium.

Výsledky testů jsou použity pro sestavení základních statistických dat, jako např. průměrná odezva serveru nebo procentuální poměr úspěšných a neúspěšných testů. Mimo tedy samotné testování funkčnosti se zaměřuje i na monitoring stavu API, a to jak po výkonnostní stránce, tak po stránce stability. Z takto nasbíraných údajů umí Runscope generovat a odesílat periodické e-maily.

Tato služba je placená, nabízí celkem čtyři možnosti. Jednotlivec má k dispozici službu v lehce omezené míře zdarma, pro malý tým (do pěti členů) stojí služba 79 $ měsíčně. Pro větší tým (do 20 členů) stojí 199 $ a pro velké společnosti pak nabízí vlastní ceník [15].

Stejně jako vREST, i Runscope podporuje více testovacích prostředí stejného API. Prostředím je myšleno oddělení např. vývojového a produkčního serveru. Podporu pro testování a monitoring rozdílných verzí API poskytuje pomocí jednotlivých prostředí (*environments*), případně pak *bucketů*. Podporu pro verzování API jsem však v systému nenašel.

Runscope nabízí vlastní API pro spojení s nástroji kontinuální integrace, propojení s IM komunikátory a mnoho jiných služeb [16]. Integraci do issue trackerů řeší pomocí služby *Zapier*, díky které dokáže propojit vlastní API až s 250 službami třetích stran [17].

Runscope nabízí opravdu plno funkcí. Pro jejich detailnější prozkoumání raději odkáži čtenáře na stránky s dokumentací (`https://www.runscope.com/docs`), ze které je poměrně dobře zřejmé, co vše aplikace dokáže.

Cílové skupiny uživatelů

Vzhledem ke stejnému nebo velmi podobnému zaměření, jako výše zmiňovaná služba vREST, i cílové skupiny uživatelů se příliš lišit nebudou. Dá se očekávat, že systém budou využívat primárně vývojáři API společně s testery.

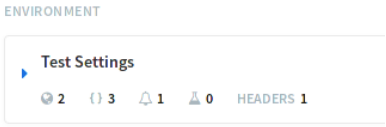
Vzhledem k tomu, že Runscope nabízí nejen testování jednorázové, ale i pomocí naplánovaných periodicky spouštěných testů, lze očekávat, že systém bude využívat člověk zodpovědný za kvalitu produktu, případně někdo podobný, který nemá přímou souvislost s vlastním vývojem. U tohoto člověka nelze ve všech případech očekávat detailní znalost terminologie případně technických detailů.

Heuristická analýza

Uživatel ví, kde se nachází, neboť aktuální stránka je vždy zvýrazněna modrým podtržením v levém menu. Po přihlášení do Runscope se uživateli zobrazí přehled všech testů. Oproti službě vREST, označení *test* zde neznamená

„množina proběhnutých a zkontrolovaných requestů na API server“, ale jedná se jednoduše o skupinu *requestů* na konkrétní API metody. Zjednodušeně lze mluvit o kategoriích. Díky tomuto přehledu uživatel ihned získá přehled o problémech, které mohly vzniknout.

Plocha celého uživatelského rozhraní je vyplněna přiměřeně. Vzhled a ovládací prvky (ikony) podporují přehlednost aplikace a cílové skupiny nebudou mít problém s porozuměním jejich významu. Jedinou negativní drobnost, kterou jsem objevil, je absence popisků ikon v nastavení testu. Pro ilustraci přikládám obrázek 2.10. Očekával bych, že přejetím kurzorem přes danou ikonu se zobrazí titulek s názvem konkrétní sekce. Kliknutí na danou ikonu ovšem již zobrazí patřičnou sekci.

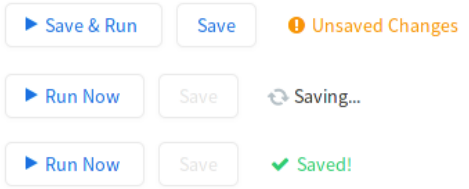


Obrázek 2.10: Runscope: Ikony nastavení testu

V případě, že uživatel udělá nějakou změnu v nastavení částí testu, zobrazí se v horní části stránky oranžový vykřičník varující o neuložených změnách. Samotné uložení změn ilustruje obrázek 2.11. Mimo informace o neuložených změnách se tato „animace“ vyskytuje u všech formulářů. Samotné spuštění testu zařadí vybrané requesty do fronty, která se zobrazuje v dolní části levého sidebaru. Nenašel jsem tedy na žádnou akci, která by neposkytla uživateli patřičnou a dobře viditelnou zpětnou vazbu.

Během analyzování Runscope jsem nenarazil na nekonzistentní pojmenování ovládacích prvků, snad jen kromě *querystring* a *URL parameter*. Na toto pojmenování jsem narazil v editaci requestu, viz obrázek 2.12. Nicméně v tomto případě se možná nejedná ani tak o nekonzistenci, jako spíš záměr návrhářů prostředí.

Během analýzy jsem se nesetkal s žádnou akcí, ze které by nebylo možné se jednoduchým způsobem dostat zpět. Tlačítka prohlížeče pro procházení historií fungují dle očekávání. Jednotlivé ovládací prvky a dostupné akce jsou viditelné dobře.



Obrázek 2.11: Runscope: změna tlačítka při uložení

Obrázek 2.12: Runscope: Querystring a URL parameter

Dokumentace je vyvedená zdařile, vždy se mi podařilo během několika málo okamžiků najít odpověď na hledaný problém. Mimo samotnou dokumentaci Runscope zasílá (volitelně) i denní přehled o aktuálním zdraví testovaného API. Mimo přehledu mi přišly i dva e-maily s nabídkou pomoci a konzultací. Otázka podpory je tedy spolehlivě pokryta.

Jak je vidět z předchozího textu, heuristická analýza v případě této služby neodhalila žádné výrazné nedostatky. I přes tento fakt mi osobně ale trvalo nějaký čas, než jsem se s uživatelským prostředím „sžil“.

2. Konzistence terminologie	
Querystring a *URL parametr*	triviální
4. Zpětná vazba	
Chybějící popisek u ikon nastavení	nízká

Tabulka 2.3: Odhalené problémy při používání Runscope

2.6 Desktopové aplikace

Testováním API se nezabývají jen webové služby, ale samozřejmě existují i instalovatelné aplikace. Jako příklad takové aplikace uvedu SoapUI. V dalších srovnáních se s tímto nástrojem již zabývat nebudu, neboť srovnávat desktopovou aplikaci s webovou službou by nebylo příliš vypovídající.

2.6.1 SoapUI

SoapUI je příkladem robustní desktopové aplikace, která nabízí (mimo testování REST API) i mnoho jiných funkcí. SoapUI disponuje nástroji na testování Soap API, mockování API serverů [5], testování bezpečnosti, load testing [6] a spoustu dalších věcí.

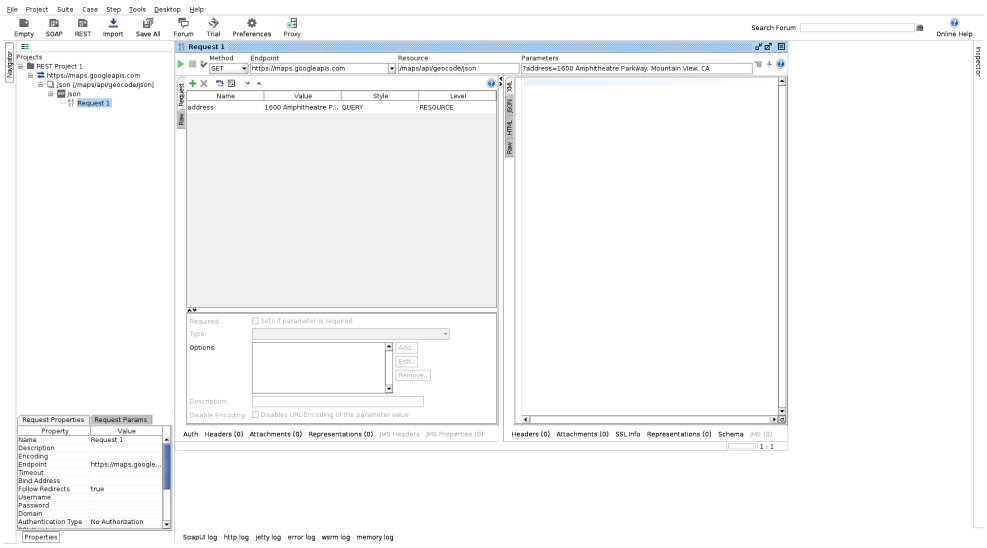
Mimo bohatou škálu nabízené funkcionality je SoapUI integrovatelný i do nástrojů pro kontinuální integraci, např. Hudson [18]. Kromě nástrojů pro kontinuální integraci lze propojit SoapUI i s několika IDE, jako například IntelliJ, NetBeans, nebo Eclipse. [18]

[5] Vytvoření metod, které pouze vypadají, jako funkční, ale slouží jen pro demostrační účely.

[6] Testování zátěže a chování aplikace pod zátěží

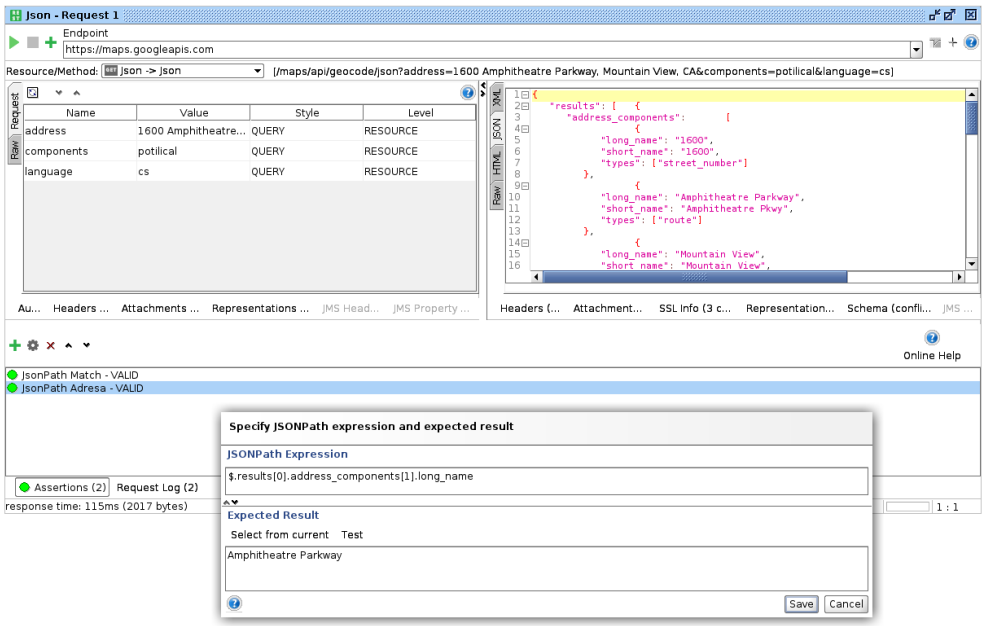
V případě, že by si uživatel nevystačil s funkcemi, které jsou nabízeny zdarma, lze zakoupit i *NG Pro* verzi, která přináší další zajímavé funkce, pro stručný přehled uvedu např. podporu datasetů pro testy, případně kontrolu pokrytí aplikace testy.

SoapUI nabízí podporu pro práci v týmu, nicméně je realizována formou sdílení konfiguračních souborů pomocí VCS [7] nástroje, což není zrovna nejvhodnější metoda. Nicméně vzhledem k povaze aplikace naprosto dostačující a pochopitelný způsob [19].



Obrázek 2.13: SoapIU: GUI aplikace

[7] Version Control System - systém pro správu revizí



Obrázek 2.14: SoapIU: detail testu

2.7 Shrnutí funkcionality

V tabulce 2.4 uvádím stručný přehled a porovnání jednotlivých funkcionalit hodnocených služeb.

2.8 Shrnutí heuristických analýz

Jak lze vidět z uvedených tabulek, Apiary a Runscope poskytují použitelné uživatelské prostředí. Měl jsem problém s používáním rozhraní služby vREST, což se také odrazilo na výsledném hodnocení. U služby Apiary jsem identifikoval jeden problém, který jsem označil závažností *střední*, u Runscope dokonce *triviální*. Poslední analyzovaná služba vREST obsahuje 4 závažné problémy, 3 středně závažné a 1 málo závažný.

2.8.1 Závěr

Jak je vidět nejenom z tabulky 2.4, vREST a Runscope tvoří poměrně zajímavé konkurenty v oblasti testování API. Plusové body získává vREST za nižší cenu, ale ztrácí v omezení na prohlížeč Google Chrome společně s nutností instalace rozšíření. Runscope vede v periodickém testování, integrací do mnoha služeb (včetně např. IM klienta Slack [20]) a zpracování uživatelského prostředí.

Je poměrně překvapivé, že ani jedna služba nenabízí responzivní verzi webu, použití na mobilním telefonu je tedy víceméně nemožné. Dále jsem

	Apiary	**vREST**	**Runscope**
Dostupnost			
Responzivní verze webu	ne	ne	ne
Podporované prohlížeče	vše	Chrome	vše
Dostupná nápověda	ano	ano	ano
Jiná lokalizace než anglická?	ne	ne	ne
Cena			
Zkušební doba	30 dní	30 dní	14 dní
Jednotlivec	zdarma	10 $	zdarma
Malý tým (5 a méně členů)	zdarma	do 50 $	79 $
Střední tým (10 a méně členů)	–	do 100 $	–
Velký tým (více než 10 členů)	99 $	od 100 $	199 $
Funcionalita			
Definice vlastního validátoru	ne	ano	ano
Mockování serverů	ano	ano	ano
Řetězení dotazů	ne	ano	ano
Podmíněné vykonávání dotazů	ne	ne	ano
Pauza mezi jednotlivými dotazy	ne	ano	ano
Definice vlastních hlaviček	ano	ano	ano
Vkládání datasetů pro běh testů	ne	ne	ne
Notifikace o výsledku testu	ne	ne	ano
Podpora verzování API	ano	ano	ne
Organizace testů do test-case	ano	ano	ano
Podpora práce v týmu			
Uživatelské role	ano	ano	ano
Podpora projektů	ano	ano	ano
Integrace			
CI nástrojů	ano	ano	ano
Project management nástroje	ne	ano	ano

Tabulka 2.4: Přehled funkcionalit existujících aplikací

očekával podporu pro načítání dat z libovolného zdroje (datasetu). Absence této funkcionality je však dána způsobem testování zadaných metod.

S přihlédnutím k funkční výbavě a přístupnosti uživatelského prostředí je z uvedených služeb jednoznačně nejlepší Runscope.

Kapitola 3

Analýza

3.1 Vymezení pojmů

Ještě před samotnou analýzou si dovolím konkrétně definovat pojmy používané nejenom v textové části této práce, ale následně i ve výsledné aplikaci. Konkrétně se jedná o tyto pojmy:

- Test
- Request
- Prostředí
- Projekt
- Verze API
- Validátor

Projekt

Tímto pojmem se myslí aplikace, API server, služba poskytující nějaké API.

Verze API

Verze API se váže ke konkrétní podobě requestu. Během vývoje aplikace nepochybně dojde k určitým změnám ve struktuře odpovědi (*response*) z API serveru. Z toho důvodu je důležité API verzovat. Pro udržení podpory je tedy potřeba testovat nejen nejnovější podobu requestů, ale i jejich podobu ve verzích minulých.

Prostředí, nebo environment

Prostředí, označované též anglicky jako *environment*, si lze představit jako konkrétní nasazení daného *projektu*, případně vývojovou větev. Důvodem, proč zavádím tento pojem (a dělení projektu), je potřeba testovat nejen vývojové, ale i testovací, potažmo produkční nasazení konkrétního projektu. S přihlédnutím k faktu, že na produkčním nasazení aplikace není vhodné např. vytvářet nové zdroje, ale je více než žádoucí kontrolovat formát a strukturu odpovědí získávaných dat.

Test

Test označuje skupinu vzájemně (ideálně souvisejících) *requestů*. V rámci testu lze předávat proměnné z jednotlivých *dotazů* (a dotazy tak lze řetězit). Jeden test může obsahovat jeden až mnoho requestů. Zároveň se jedná o nejmenší testovatelnou jednotku v systému. Samotný test pak může být v systému spuštěn buď jednorázově uživatelem, nebo periodicky v určitých intervalech.

Test je splněn, pokud všechny jeho dotazy odpovídají specifikaci definované pomocí *validátorů*.

Dotaz, nebo request

Dotaz (*request*) označuje jedno volání konkrétní metody na API serveru. K jednomu requestu lze přiřadit jeden nebo více *validátorů*, které rozhodnou, zda request má správný formát a splňuje všechny náležitosti.

Validátor, nebo assertion

Validátor je souhrnné označení pro rozhodovací pravidlo, zda konkrétní část odpovědi z API metody (requestu) splňuje očekávané vlastnosti. Těmito vlastnostmi může být jak existence konkrétního elementu v struktuře odpovědi, tak i maximální čas potřebný pro vykonání requestu, nebo přítomnost předem určených HTTP hlaviček (headers).

3.2 Identifikace uživatelských požadavků

Pro účely identifikace požadavků uživatelů, resp. pro zjištění, co uživatel očekává od aplikace zabývající se testování REST API jsem oslovil celkem 5 lidí ze svého okolí. Jednalo se jednak o studenty-spolužáky a jednak kolegy z praxe. Oslovoval jsem jen ty lidi, o kterých jsem věděl, že mají určitou zkušenost buď s programováním aplikací konzumujících nějaké API, a nebo se na tvorbě API sami podíleli. Požadavky jsem sbíral vždy formou interview.

Hlavním uživatelským cílem je určitě samotné testování a s tím spojené jednoduché *vytvoření nového testu*, resp. *přiřazení dotazu do testu*. Dle do-

stupných odpovědí uživatelé více ocení jednoduché vložení a následné detailní úpravy, než vyplňování rozsáhlého formuláře.

Další důležitou funkcí, kterou uživatel od aplikace očekává, je zobrazení samotného detailu výsledku. V tomto detailu uživatel očekává stručné statistiky (průměrná doba odpovědi, velikost, procentuální úspěšnost dotazů), samozřejmě seznam proběhnutých dotazů společně s jejich výsledky. Výsledky chce mít uživatel k dispozici i zpětně, tedy u testu není možné ukládat jen poslední výsledek.

Ukázalo se, že uživatelé mají zájem o hlubší organizaci testů, než jen na úrovni projektů. V drtivé většině případů výsledná aplikace neběží jen v jednom prostředí. Naopak existuje např. jedna verze pro vývoj a druhá verze pro produkční nasazení. Kvůli tomuto problému se dotazovaní shodli na potřebě rozdělit samotný projekt do více „kategorií“, ve své práci to označuji také jako „prostředí“.

Vzhledem k faktu, že většina aplikací stavěných nad API vyžaduje autentifikaci, nástroj na testování by měl podporovat volání i neveřejných metod.

Uživatelé chtějí pracovat v týmu, kde každý uživatel má svojí roli a je nějakým způsobem limitován. Jeden z respondentů přišel na zajímavý nápad s rolí uživatele, který má přístup do systému jen pro čtení, nemůže tedy měnit nastavení, přidávat testy, ani samotné testy spouštět.

Důležitým požadavkem je, aby aplikace poskytovala dostatečný přehled o stavu testovaných dotazů a v co možná nejkratším čase oznámila uživateli nově vzniklý problém.

3.2.1 Závěr

Tento drobný průzkum mi jednak potvrdil mé vize o funkcionalitě aplikace, ale zároveň vyvrátil mé představy o procesech v jejím používání. Původně jsem předpokládal, že uživatel bude vytvářet strukturu API v podobě definice zdrojů, metod a seznamu všech vstupních parametrů. Test by pak byl tvořen jednak definovaným dotazem a jednak hodnotami vstupních parametrů, z nichž ne všechny by musely být povinné. Tento způsob tvorby testů se však všem dotazovaným osobám zdál komplikovaný a zdlouhavý. Dále mě překvapilo, že pro uživatele není prioritou dokonalé zobrazení na mobilním telefonu.

Tato sebraná data poslouží jako základ pro sestavení funkčních a nefunkčních požadavků realizované aplikace.

3.3 Funkční požadavky

3.3.1 Organizace testů do projektů

Uživatel bude vytvářet projekty, což bude základní organizační jednotka. U projektů bude ukládáno jeho jméno a popis pro lepší přehlednost v systému.

3.3.2 Podpora testovacích prostředí

Projekt bude dělen do tzv. prostředí. Prostředí tvoří základní organizační jednotku v rámci jednoho projektu. Jeden projekt může obsahovat více prostředí, ale jedno prostředí patří právě jednomu projektu. Do informací o prostředí bude ukládáno jméno, popis a URL adresa testovaného API serveru. Tyto informace budou sloužit jen pro zpřehlednění aplikace.

3.3.3 Podpora týmové spolupráce

Pro podporu týmové spolupráce budou v aplikaci implementovány jednotlivé role pro uživatele. Tyto role budou následující: *manažer*, *editor* a *host*. Konkrétní odpovědnosti a možné akce viz use-case diagram na obrázku 3.1.

Uživatelská role je definována na úrovni prostředí, tedy pro každé prostředí definované v projektu může mít uživatel jinou roli.

3.3.4 Testování odpovědí ve formátu JSON

Aplikace bude testovat odpovědi z API serveru ve formátu JSON.

3.3.5 Autentizace požadavků

Aplikace bude podporovat i testování API metod, které jsou přístupné jen po přihlášení. Aplikace bude umožňovat následující způsoby přihlášení:

- Basic Authorization
- JSON Web Token (Bearer token)

3.3.6 Podpora HTTP hlaviček

Systém bude umožňovat definici HTTP hlaviček pro projekt, prostředí, test a požadavek. Tyto HTTP hlavičky se budou odesílat při každém požadavku odeslaného prostřednictvím aplikace.

3.3.7 Pravidla pro kontrolu struktury odpovědi

Základem aplikace bude validace struktury odpovědí na API požadavek. Systém bude umožňovat přidání *validátoru* k requestu. Kontrolovány budou následující kategorie:

- Status HTTP odpovědi
- Ne/existence klíče/parametru
- Hodnota klíče/parametru

- Počet prvků
- Obsah těla odpovědi

Ke každé kategorii bude možnost definovat „operátor“, tedy jedno z následujících:

- Obsahuje
- Neobsahuje
- Rovná se
- Nerovná se
- Menší než
- Menší nebo rovno
- Větší než
- Větší nebo rovno

3.3.8 Výkonnostní pravidla validace requestu

Systém bude podporovat i validaci požadavku na základě rychlosti odezvy serveru. Pro tuto kontrolu bude možné využít stejných operátorů porovnání jako u pravidel pro kontrolu struktury odpovědi.

3.3.9 Historický přehled výsledků testů

Ke každému proběhnutému testu bude ukládána historie requestů, odeslaných HTTP hlaviček, odpovědi a přijatých HTTP hlaviček. Mimo tato surová data bude historický přehled obsahovat i seznam validátorů společně se stavem, jak jejich vyhodnocení dopadlo. Každý výsledek bude mít svojí vlastní URL adresu.

3.3.10 Statistické údaje

Z výsledků testů, a to jak kvalitativních (kontrola struktury odpovědi), tak i výkonnostních (rychlost odezvy), budou zobrazovány statistické údaje. Tyto statistické údaje budou zobrazovány za posledních 24 hodin, poslední týden (7 dní) a měsíc (30 dní).

Ze statistických údajů nás zajímá počet testovaných požadavků, průměrná a nejdelší doba odezvy serveru a procentuální úspěšnost testovaných požadavků.

3.3.11 Plánování spuštění testů

Pro každý test bude umožněno zvolit datum a čas, kdy se má spustit.

3.3.12 Periodické spouštění testů

Aplikace bude podporovat nejenom jednorázové, ale i periodické spouštění testů. Intervaly pro spouštění testů jsou následující:

- Každých 30 minut
- Každou 1 hodinu
- Každých 6 hodin
- Každý den

V případě, že test má zároveň i naplánovanou dobu spuštění, první spuštění proběhne v tuto zvolenou dobu.

3.3.13 Odesílání URL parametrů

Systém bude poskytovat správu URL parametrů (tzv. *query string* parametry) pro jednotlivé dotazy. Tyto parametry budou specifikovány rovnou jako součást URL adresy requestu (/resource/method?**foo=bar**).

3.3.14 Odesílání HTTP parametrů

Kromě URL parametrů bude aplikace umožňovat i testování requestů, které vyžadují parametry předávány jiným způsobem než v URL adrese. Těmito parametry se myslí POST, PUT a PATCH parametry požadavku.

3.4 Nefunkční požadavky

3.4.1 Interaktivní uživatelské prostředí

Aplikace bude stavěna jako tzv. Single-page aplikace a díky tomu bude nabízet interaktivní prostředí nevyžadující znovunačtení stránky po každé uživatelské akci.

3.4.2 Přístupné uživatelské prostředí

Aplikace musí mít přístupné uživatelské prostředí. Přístupnost bude hodnocena na základě Nielsenovy heuristické analýzy provedené v závěru implementace funkčního prototypu.

3.4.3 Vícejazyčnost

Aplikace bude primárně vyvíjena v českém jazyce, ale bude podporovat snadné přidání nového jazyku. Nový jazyk bude přidáván jako katalog překladů.

3.4.4 Výkonnost

Získání dat ze serveru s následující konfigurací musí být vždy nižší než 1s. Měření bude provedeno pomocí nástroje `ab` se třemi konkurenčními požadavky.

- CPU: Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
- RAM: 2 GiB

3.5 Volba technologií

Správná volba technologií dokáže vývoj nejen urychlit, ale zároveň i přispět k budoucí rozšiřitelnosti. A na základě zvolené technologie mohou vyplynout některé části analýzy. Příkladem budiž rozdílnost databázového modelu pro SQL databázi v porovnání s modelem pro NoSQL databázi.

Při implementaci výsledné aplikace se chci seznámit s jinými technologiemi, než které jsem do této doby používal. Pro své webové aplikace jsem používal PHP společně s MySQL databází. Pro tuto práci bych se chtěl seznámit s jiným přístupem ke tvorbě interaktivních webových aplikací.

3.5.1 Backend

Mimo uživatelské prostředí a úložiště dat je potřeba ještě zvolit technologii, ve které bude implementována vnitřní logika celé aplikace. Pro tento účel volím mezi třemi technologiemi - Ruby, Python a Node.js.

3.5.1.1 Ruby

Ruby je dynamicky typovaný, intepretovaný jazyk vyvíjený již od roku 1995 [21]. Velmi zajímavou vlastností je, že vše – včetně primitivních datových typů – jsou objekty.

Vzhledem k faktu, že budu vyvíjet webovou aplikaci, je nasnadě sáhnout po hotovém frameworku určeném pro vývoj právě webových aplikací. Tímto frameworkem v případě Ruby je např. známý Ruby on Rails (`http://rubyonrails.org/`). Jak se lze dočíst na oficiálních stránkách projektu, tento framework byl poprvé uveřejněn roku 2004. Jedná se tedy o vyspělý a odladěný kus softwaru.

Mnoho velkých projektů je postaveno nad Ruby on Rails, například GitHub, Redmine, Airbnb a mnoho jiných [22, 23, 24].

3.5.1.2 Python

Python je jazykem s velkou tradicí. První vydání spatřilo světlo světa již kolem roku 1980 [25]. Jedná se vskutku o všestranný jazyk, píší se v něm skripty pro obsluhu serverů, aplikace s grafickým rozhraním, webové aplikace, nebo obsluhuje mikroprocesory (viz projekt MicroPython – `https://micropython.org/`).

Za dobu své existence existuje velké množství nejrůznějších knihoven a frameworků. Pro účely vývoje webu bych zvolil framework Django (`https://www.djangoproject.com/`), který přináší vše, co programátor od webového frameworku očekává - správu překladu (lokalizace), zabezpečení proti CSRF, šablonovací systém, autentifikaci uživatelů, podporu migrací, ... [26].

Stejně jako Ruby on Rails, i Python Django se používá u mnoha velkých projektů. Například Instagram, Pinterest nebo Mozilla [27, 26].

3.5.1.3 Node.js

Node.js není nic jiného, než Javascript interpretovaný na serveru pomocí enginu V8. Jedná se o *event-driven* jazyk postavený nad neblokujícími I/O operacemi. Díky neblokujícím voláním a asynchronnosti jsou aplikace psané v Node.js velmi rychlé.

Pro vývoj aplikací v Node.js bych zvolil nějaký MVC framework. Při průzkumu možností mě nejvíce zaujal framework Sails.js (`http://sailsjs.org/`.

Node.js je poměrně mladým jazykem, první verze byla představena roku 2009 na evropské konferenci JSConf [28]. Node.js používán v mnoha velkých aplikacích, například Uber [29], nebo Netflix [30].

3.5.1.4 Závěr

Jak známo, „nejlepší“ jazyk neexistuje. Pro účely této práce jsem se rozhodl zvolit Node.js, proto, že se jedná o poměrně mladou technologii, na kterou již ale přešla řada společností. Zároveň nabízí zajímavý výkon, snadnou škálovatelnost a sjednocuje jazyk použitý na klientské části aplikace s jazykem použitým v backendu aplikace. V neposlední řadě si myslím, že vzhledem k celkovému charakteru aplikace bude díky svojí asynchronnosti vhodnější volbou, než ostatní uvedené jazyky.

3.5.1.5 Sails.js

Výběru frameworku pro vývoj aplikace v Node.js jsem nevěnoval tolik času, jako výběru samotného jazyka, neboť jsem příliš nevěděl, co od technologie konkrétně očekávat. Místo toho jsem si zkusil nainstalovat a zprovoznit jednoduchou aplikaci (v zásadě se jednalo o výpis dat z databáze) a na základě této zkušenosti učinil finální rozhodnutí.

Vybíral jsem z frameworků Sails.js, Koa a total.js. Z těchto tří frameworků se mi nejlépe pracovalo s prvním uvedeným, tedy Sails.js a proto jsem se rozhodl ho využít pro implementaci celé aplikace. Tento framework obsahuje router (Express.js), ORM (Waterline), podporuje řízení přístupu k jednotlivým metodám pomocí tzv. *policies*, obsahuje v sobě knihovnu pro práci s WebSockety a v neposlední řadě nástroj zvaný Grunt [31].

Grunt slouží ke správě opakujících se úloh, kterých je při vývoji aplikace poměrně velké množství. Jedním z příkladů může být vkládání nového skriptu do stránky pomocí tagu `<script>`, případně minifikace[8] jak javascriptových skriptů, tak CSS souborů.

3.5.2 Databáze

Aplikace bude uchovávat dva typy dat - strukturovaná, vhodná pro relační zpracování a nestrukturovaná. Strukturovaná data představují informace o projektech, uživatelích, příslušnosti requestů k testům atp. Naopak data nestrukturovaná představují vlastní volání requestů a odpovědi na ně. Pro nejen statistické účely je potřeba uchovávat data i historicky, z toho důvodu lze očekávat, že velikost databáze se bude neustále zvětšovat.

3.5.2.1 Mongo DB

Mongo DB je představitelem NoSQL dokumentově orientované databáze, která vznikla relativně nedávno, teprve roku 2009. Už v základu je navržena pro distribuovaný běh, horizontální škálování nachází podporu rovnou v návrhu celé databáze. Slibuje vysoký výkon a to jak při zápisu, tak při čtení.

MongoDB používá mnoho velkých společnostní i malé startupy, jako příklad uvedu Foursquare, Adobe, Bosch, LinkedIn a mnohé další [32].

3.5.2.2 PostgreSQL

Ve své praxi jsem se setkal převážně s databází MySQL. Vzhledem k faktu, že jedním z mých cílů v této práci je i naučit se novou technologii, rozhodl jsem se MySQL nevyužít a místo toho do úvahy vzít jedno z konkurenčních řešení, konkrétně PostgreSQL. Srovnáním MySQL a PostgreSQL se zabývá mnoho článků, uvedu např. `https://www.wikivs.com/wiki/MySQL_vs_PostgreSQL`.

PostgreSQL je, stejně jako Mongo DB, využíváno v mnoha velkých projektech. Seznam společností využívajících tuto databázi lze najít na adrese `http://www.postgresql.org/about/users/`

3.5.2.3 Závěr

Pro účely této práce jsem se nakonec rozhodl použít relační databázi PostreSQL. Jedním – pravděpodobně tím hlavním – z důvodů je fakt, že s relační

[8] odstranění zbytečných částí kódu jako např. komentáře a prázdné řádky

databází mám největší zkušenosti. Díky tomu věřím, že se mi povede lépe sestrojit databázový model a tím pádem i celou databázi pro aplikaci. Mimo to PostgreSQL nabízí mnoho zajímavých funkcí, které MySQL neumí, např. propracovanější indexování sloupců. Ukládání requestů a odpovědí vyřeším pomocí datového typu JSON. Posledním důvodem, proč nechci použít Mongo DB, je obava z množství nových technologií. Přeci jen návrh SQL a NoSQL databází se dosti liší a samotný backend hodlám vyvíjet v Node.js, se kterým mám jen velmi malé zkušenosti. Upgrade aplikace, resp. rozštěpení databází na dvě části (tedy relační a dokumentově orientovanou) nechávám pro případné přispěvatele do projektu jako námět k vylepšení.

Přestože PostgreSQL není primárně clusterová databáze, určité možnosti škálování nabízí. Detaily, resp. dostupné metody škálování lze nalézt v oficiální Wiki dokumentaci [33]. Na jednom uzlu je dle několika měření PostgreSQL rychlejší [34, 35], což ovšem neznamená, že při přidání více uzlů tomu tak bude stále. PostgreSQL a Mongo DB jsou dvě odlišné databáze a každá se hodí na jinou věc.

3.5.3 Uživatelské prostředí

Nástrojů ulehčujících práci při tvorbě uživatelského prostředí existuje nespočet. Větší aplikace navíc nevyužívají jen jednu knihovnu, ale takřka výhradně vznikají nějakou kombinací. Pro účely vývoje jsem se rozhodl postavit uživatelské prostředí na responzivním frameworku Foundation (`http://foundation.zurb.com`) s využitím Flex grid systému. Tento framework zajišťuje (resp. usnadňuje) tvorbu responzivního uživatelského prostředí. Samotnou interakci se serverem však neposkytuje. Pro tento účel jsem zvolil knihovnu AngularJS.

3.5.3.1 ZURB Foundation

Jak jsem uvedl v textu týkajícího se volby technologií, rozhodl jsem se použít HTML framework Foundation 6, který nabízí jednodušší přístup ke tvorbě (nejen) responzivního layoutu aplikace. Podobně jako „konkurenční“ Bootstrap, tak i Foundation je sada CSS (resp. SASS) stylopisů a javascriptových skriptů. Javascript je závislý na knihovně jQuery, kterou implementovaná aplikace využívá víceméně jen pro účely Foundation frameworku.

- Dokumentace: http://foundation.zurb.com/sites/docs/

3.6 Volba úložiště zdrojových kódů

Kód budu verzovat pomocí nástroje Git, ve kterém budu pracovat se dvěma remoty. Jeden remote si nastavím na svém serveru, kde aplikace následně i

poběží. Na tomto serveru mám k dispozici instalaci issue trackeru Redmine, do kterého chci mít repozitář integrovaný.

Druhý repozitář bude obsahovat totožný obsah, jen bude dostupný veřejnosti pouze pro čtení.

Pro účely své práce jsem se rozhodl využít službu **GitHub** (`https://github.com/`), a to i přes menší úložiště a o něco menší možnosti správy repozitáře a kódu. Hlavním důvodem mého rozhodnutí jsou aplikace integrovatelné do repozitáře. Vzhledem k tomu, že aplikaci vydávám jako open-source, budu mít tak možnost si vyzkoušet i jiné služby, např. zmiňovaný Travis CI.

Primárně budu používat Git nainstalovaný na svém serveru, na kterém i aplikace poběží. GitHub budu využívat jen v podobě dalšího *remote*. V případě potřeby není problém přidat nový remote a začít používat jinou službu.

3.7 Use case diagram

Na základě sebraných požadavků od uživatelů uvedených v části 3.2 jsem identifikoval jednak role uživatelů a jednak jednotlivé případy použití aplikace. Případy užití ilustruje obrázek 3.1.

3.7.1 Identifikace typů uživatelů

Celkem jsem identifikoval pět typů uživatelů. Schválně zde nemluvím o rolích, neboť toto označení používám v souvislosti s právy uživatele v rámci prostředí. Typy uživatelů jsou následující:

3.7.1.1 Nepřihlášený uživatel

Nepřihlášený uživatel je libovolný návštěvník stránky. V systému nemá prakticky žádné oprávnění, jediné dvě akce, které smí provádět, je přihlášení a registrace nového účtu.

Autor projektu

Autor projektu je uživatel, který vytvořil projekt. V rámci vytvořeného projektu má plnou správu nad jeho nastavením a smí vytvářet a mazat nová prostředí. V případě, že je z nějakého prostředí odstraněn jiným manažerem, může být zpět do prostředí přiřazen s jinou, klidně i méně privilegovanou rolí, než je *manažer*.

Manažer

Tato uživatelská role má na starosti kompletní správu prostředí, do kterého byl přiřazen. V okamžiku vytvoření prostředí se uživatel stává automaticky jeho manažerem a má přístup do správy uživatelů a nastavení prostředí. Manažer

může vykonávat veškeré akce, které mohou vykonávat uživatelé v roli *Tester*, případně *Host*.

Tester

Tester je opět uživatelská role, která umožňuje správu testů a požadavků. Oproti roli *Manažer* nemá Tester právo měnit nastavení prostředí ani spravovat uživatele daného prostředí.

Host

Host je nejméně privilegovaná uživatelská role. Do prostředí má přístup jen pro prohlížení výsledků testů (a tím pádem i k zobrazení všech testů v konkrétním prostředí). Nemůže měnit nastavení samotného prostředí, testů ani požadavků, stejně jako testy nemůže spouštět a ani plánovat jejich spuštění.

3.7.2 Seznam případů užití aplikace

3.7.2.1 Registrace

Aby bylo možné aplikaci začít používat, je potřeba vytvořit uživatelský účet. Registrovat se může kdokoliv, dokonce i pokud je již přihlášený.

3.7.2.2 Přihlášení

Pro používání veškerých komponent aplikace je vyžadováno přihlášení. Přihlásit se může libovolný uživatel a to dokonce i tehdy, je-li již přihlášen (např. pokud chce změnit účet). Pro účely samotného přihlášení nemusí být uživatel odhlášen.

3.7.2.3 Prohlížení seznamu testů

Umožňuje prohlížet seznam všech vytvořených testů v daném prostředí. Tento seznam může zobrazit libovolný uživatel, který má k prostředí přístup (byl k němu přiřazen Manažerem prostředí).

3.7.2.4 Zobrazení historie výsledků testů

Zobrazí historii proběhnutých testů na základě vybraného testu. Historii může zobrazit libovolný uživatel, který má k prostředí přístup (byl k němu přiřazen Manažerem prostředí).

3.7.2.5 Naplánování spuštění testu

Zvolí čas, kdy bude test spuštěn. Naplánování spuštění testu je druh editace testu, je proto zapotřebí uživatelská role *Manažer* nebo *Tester*.

3.7.2.6 Spuštění testu

Zařadí test do fronty na vyhodnocení testu. Test může spustit pouze *Manažer* nebo *Tester*.

3.7.2.7 Správa testů

Správa testů je souhrnné označení pro vytvoření a editaci informací o již existujícím testu. Je vyžadována role *Manažer* nebo *Tester*.

3.7.2.8 Správa requestů

Ve správě requestů (dotazů) je zahrnuto jejich vytváření, editace, HTTP parametrů a HTTP hlaviček. Je vyžadována role *Manažer* nebo *Tester*.

3.7.2.9 Správa validátorů

Správa validátorů obsahuje přiřazení, editaci a odstranění validátoru. Je vyžadována role *Manažer* nebo *Tester*.

3.7.2.10 Přidání requestu do testu

Přiřadí již vytvořený request k existujícímu testu. Je vyžadována role *Manažer* nebo *Tester*.

3.7.2.11 Úprava nastavení prostředí

Umožní změnu informací o prostředí a nastavení HTTP hlaviček, které budou použity pro všechny dotazy v daném prostředí. Je vyžadována role *Manažer*.

3.7.2.12 Úprava nastavení projektu

Umožní změnu informací o projektu a nastavení HTTP hlaviček, které budou použity pro všechny dotazy v daném prostředí. Tuto akci smí vykonávat jen *Autor projektu*.

3.7.2.13 Správa uživatelů v prostředí

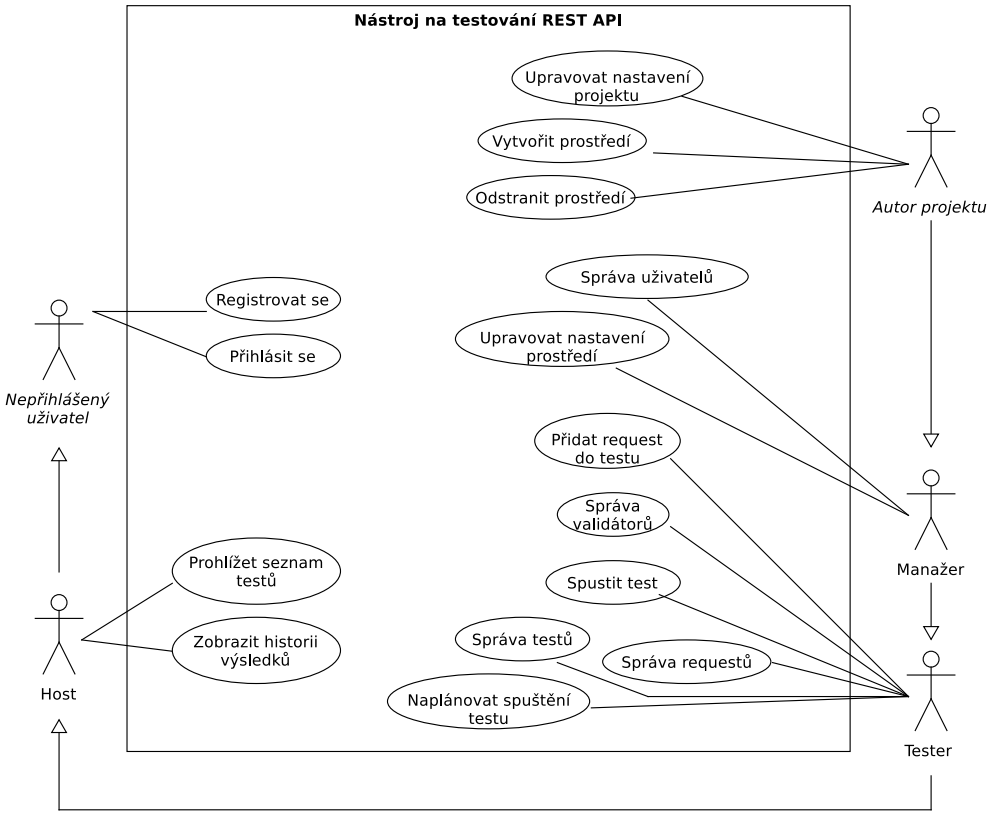
Přiřadí nového uživatele s vybranou rolí do prostředí, případně odstraní již existujícího uživatele a zakáže mu tak přístup do prostředí. Tuto akci smí provádět jen *Manažer* konkrétního prostředí.

3.7.2.14 Smazání prostředí

Odebere prostředí z vybraného projektu společně se všemi daty. Smí provést jen *Autor projektu*.

3.7.2.15 Vytvoření prostředí

Vytvoření nového prostředí ve vybraném projektu. Smí provést jen *Autor projektu*.



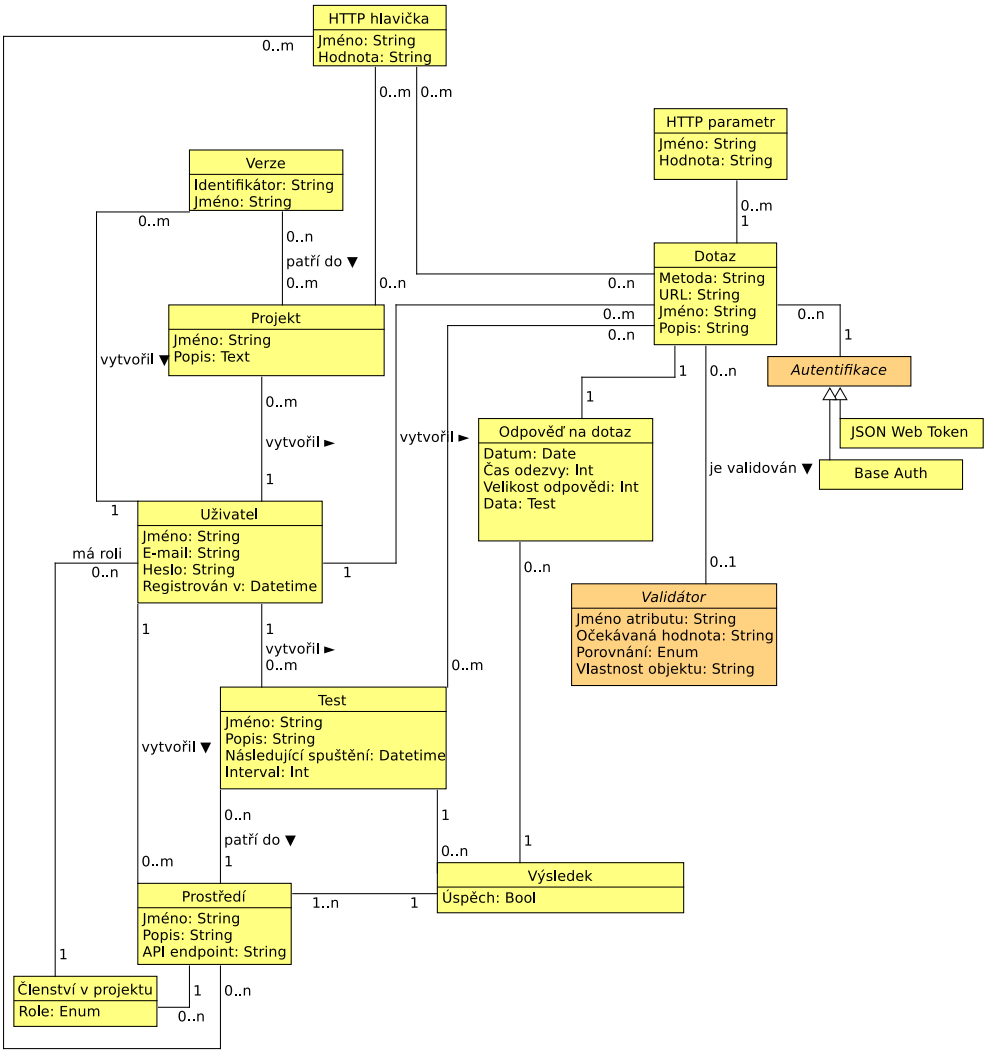
Obrázek 3.1: Diagram případů užití aplikace

3.8 Doménový model

Na základě funkčních a nefunkčních požadavků jsem zpracoval doménový model, ve kterém jsem se snažil zachytit všechny důležité vazby mezi objekty. Doménový model ilustruje obrázek 3.2

3.9 Model databáze

Databázový model jsem kreslil pomocí volně dostupného programu pgModeler (`http://pgmodeler.com.br/`). K dispozici je zdarma pod licencí GNU GPL 3 a nabízí jak tvorbu ER diagramů, tak i následný export do DDL pomocí forward engineeringu.



Obrázek 3.2: Doménový model

Pro návrh databázové struktury jsem vycházel z doménového modelu, avšak přihlédl jsem rovnou i k finální implementaci. Z tohoto důvodu obsahuje návrh velké množství vazeb na prostředí a na uživatele.

Bohužel vzhledem k výše zmíněným vazbám *entita-uživatel* a *entita-prostředí* by bylo uvedení modelu na stránkách této práce velmi nepřehledné, z toho důvodu přikládám model na CD, které je součástí práce. Na přiloženém médiu lze najít zdrojová data pro použití v programu pgModeler.

3.10 Závěr

Součástí návrhu komplexnější aplikace by jistě měl být ještě diagram tříd. V této práci ho však záměrně neuvádím, neboť struktura aplikace je dána jednak samotným frameworkem a jednak Javascript jako takový příliš dobře nepodporuje např. dědičnost, tudíž by byl diagram spíše rozšířenou kopií doménového modelu.

V této části jsem dokončil analýzu a sběr požadavků na funkcionalitu aplikace a mohu se přesunout na návrh uživatelského prostředí a následně na implementaci funkčního prototypu.

Kapitola 4

Implementace

4.1 Wireframy

Pro vytvoření wireframů jsem se rozhodl nepoužít žádný konvenční nástroj primárně určený pro tvorbu wireframů, ale rozhodl jsem se sáhnout rovnou po nástroji, který hodlám použít pro vývoj uživatelského rozhraní aplikace. Konkrétně se jedná o HTML5 framework Foundation.

Vzhledem k povaze samotných wireframů je do práce nevkládám jako obrázky, nicméně jsou dostupné na přiloženém médiu v adresáři `wireframe`, případně na adrese `http://rest-wf.kvacek.cz`.

4.1.1 Testování wireframů

Wireframy jsem se rozhodl testovat metodou obsluhované makety. Uživatelské prostředí posuzovali tři experti – kolegové z praxe. Dva pracují jako vývojáři backendu (API) a jeden se specializuje na aplikace postavené na frameworku AngularJS, který pro svojí funkci REST API vyžaduje.

Samotné wireframy jsou tvořené jako statické HTML stránky, tudíž není možné úplně dobře posoudit rychlost odezvy, proto jsem se touto částí heuristické analýzy při testování nezabýval. Další otázkou, kterou jsem neřešil, je otázka nápovědy - pro reálné nasazení se jistě jedná o nepostradatelnou část aplikace, nicméně pro účely této práce žádná uživatelská nápověda vypracována nebyla. Ve wireframech je dokumentace naznačena na titulní straně pod seznamem projektů.

Maketu jsem obsluhoval tak, že jsem postupně skrýval a odkrýval jednotlivé části HTML kódu pomocí komentářů. Po každém takovém zásahu si expert musel stránku znovu načíst, nicméně byl poučen, že takto se výsledná aplikace chovat nebude.

Podobně jako u hodnocení existujících služeb, i v případě posuzování této makety jsem nalezené problémy rozdělil do kategorií podle závažnosti. Tyto

závažnosti jsou následující: *triviální*, *malá*, *střední*, *vysoká* a *znemožňující používání*.

4.1.1.1 Heuristická analýza

Experti se shodli, že zobrazení aktuálního stavu aplikace pomocí drobečkové navigace je dostačující a díky tomu není problém s orientací v aplikaci. Dále se shodli na tom, že aplikace nepotřebuje explicitní navigaci (menu), neboť ovládací prvky jsou umístěny u konkrétních prvků a jsou tak vždy dostupné. Menu samotného – umístěného v pravé horní části záhlaví – si dva z nich vůbec nevšimli.

V otázce terminologie žádný z expertů nezaznamenal výrazný problém. Jen jeden expert se pozastavil nad termínem „Request endpoint“ použitý v editoru požadavku. Toto označení označil za „lehce matoucí“, neboť se mu zdálo, že pojem koliduje s „API endpoint“, který se používá v nastavení prostředí. Po konzultaci se zbývajícími dvěma dotazovanými vyšlo najevo, že lepším označením by bylo „API metoda“. Jiný problém nebyl zaznamenán.

Problém, který objevili všichni dotazovaní a který všichni shledali jako velmi závažný problém, byla absence zpětné vazby v podobě chybové hlášky. Při návrhu uživatelského prostředí jsem tuto otázku nezohlednil a ani nikde nenaznačil zobrazení chybové hlášky a byla to chyba. Mimo zobrazování chyb se dva experti shodli na tom, že postrádají i zpětnou vazbu o uložení (editaci) formuláře. Mimo tyto nedostatky v návrhu chyběla místy i informace o chybějících datech (nedefinované žádné HTTP hlavičky, prázdný seznam query string parametrů).

Na rozdíl od zobrazování zpětné vazby nikdo z expertů neměl problém s orientací v aplikaci ve smyslu vracení se na předešlou obrazovku, případně přechody mezi stavy aplikace. Nikdo neshledal žádný problém v nutnosti opakovat dlouhou sekvenci akcí, aby mohl nějakou akci buď opustit, nebo zopakovat. Stejně tak nikdo nereportoval problém s tlačítky prohlížeče pro procházení historie. Naopak všichni tři experti se shodli, že v okně s editorem testu očekávali i možnost nastavení jednak periodického spouštění testu a jednak naplánování příštího spuštění.

Akce, uspořádání ovládacích prvků a jejich význam bylo hodnoceno jako téměř konzistentní. Mimo rozdílné velikosti a zarovnání pár tlačítek (např. pro přidání HTTP hlaviček v nastavení projektu a v nastavení dotazu) nebyly odhaleny žádné větší nedostatky. Ohledně rozložení aplikace měli experti ovšem výhrady. Dvěma z nich přišel matoucí rozdílný layout v seznamu projektů a nastavení projektu (bez bočního panelu) a ostatních obrazovkách (s bočním panelem). Jednomu expertovi přišel zbytečný přehled o procentuální úspěšnosti testu v uvedených období. Zbylí dva respondenti se k této otázce nevyjádřili, nicméně po zralejší úvaze jsem usoudil, že při větším počtu testů by toto mohlo působit spíše nepřehledně a rozhodl se tento detail odstranit.

Místo toho jeden z hodnotitelů navrhl zobrazení spíše informace o příštím spuštění testu.

Jedinou akcí, ke které měli všichni hodnotící velké výhrady, bylo zobrazení detailu po kliknutí na výsledek testu v levém bočním panelu. Všichni bez rozdílu očekávali, že se zobrazí konkrétní výsledek testu a ne jeho detail.

Mimo výše uvedených problémů byl nalezen ještě jeden, který ovšem nespadá do uvedených devíti kategorií je otázka nápovědy. Tento problém zde uvádím i přes skutečnost, že jsem v úvodním textu napsal, že tato část hodnocena nebude. Jedná se o absenci popisu formátu, kterým uživatel může specifikovat testovanou vlastnost objektu. Nikdo z dotazovaných netušil, v jakém formátu má zadat požadovanou hodnotu. Experti se však shodli na tom, že by pro tento účel stačilo doplnit do textového políčka tzv. *placeholder* text, který by ilustroval očekávaný formát.

Poslední hodnocenou částí je využití prostoru v návrhu uživatelského prostředí aplikace. V této otázce nikdo neměl výraznější námitku, pouze jeden hodnotitel označil jako velmi triviální problém zobrazení neúspěšných testů v přehledu prostředí. Podle něj není zobrazení jednoho testu na řádek úplně ideální, neboť není využita druhá polovina panelu prostředí. V rámci zlepšení navrhl nepovedené testy neřadit do řádků, ale zobrazovat je vedle sebe.

Nalezené problémy jsem shrnul v tabulce 4.1. Číslo nadpisu každé skupiny buněk odpovídá i číslu heuristiky uvedené výše.

4.1.1.2 Závěr

Z hodnocení wireframů vyplynulo pár změn, které začlením do funkčního prototypu. Vzhledem k faktu, že testování neprobíhalo nad reálným systémem, nelze ho považovat za finální a bylo by naivní předpokládat, že po implementaci funkčního prototypu nevzniknou nové připomínky, případně problémy v přístupnosti.

V této fázi analýzy nevznikly vyloženě protichůdné názory na nějaký problém. Tento fakt připisuji tomu, že hodnotitelé spolupracují na denní bázi v rámci praxe a v otázkách uživatelského prostředí mají poměrně podobné názory. Případně i tím, že používáme v rámci firmy stejné systémy a nástroje, tím pádem může být pohled na uživatelské prostředí trochu zkreslen.

4.2 Implementace funkčního prototypu

Jak jsem uvedl v předchozím textu, pro implementaci aplikace jsem se rozhodl použít framework Sails.js postavený na Node.js a databázi PostgreSQL. Na frontendovou část jsem pak zvolil AngularJS a HTML framework Foundation. Při vývoji jsem však narazil na některé omezení, které se pokusím v následujícím textu blíže popsat.

4.2.0.1 Výměna ORM

Jak jsem uvedl v textu o Sails.js, framework samotný nabízí předinstalované ORM s názvem Waterline. To nabízí jednotné API pro přístup jak k SQL tak i NoSQL databázím pomocí různých databázových *driverů*. ZDROJ.

Při implementaci jsem však poměrně brzy narazil na omezení, které skýtá vestavěné ORM Waterline. Tímto problémem byla prakticky nemožnost integrace s existující strukturou databáze. Ani přes nemalou snahu se mi nepodařilo přinutit Waterline k vytvoření M:N vazby mezi dvěma tabulkami. Problém byl v definici jména vztažné tabulky (tabulky obsahující identifikátory spojených záznamů). Ve chvíli, kdy jsem se podíval do logu z PostgreSQL databáze, zjistil jsem, že navíc Waterline generuje velmi neefektivní SQL dotazy. Např. tam, kde by stačil dotaz s jedním `LEFT JOIN` Waterline spouštěl dotaz slučující výsledky ze dvou dotazů pomocí `UNION`. Posledním důvodem, proč jsem se rozhodl Waterline opustit je dokumentace. Málokdy se mi podařilo v ní nalézt řešení problému.

Řešením tedy bylo buď vyměnit celý framework, nebo se pokusit o výměnu jen samotného ORM. Naštěstí jsem poměrně brzy narazil na projekt zvaný Sequelize (`http://docs.sequelizejs.com/en/latest/`), který mimo poměrně kvalitní dokumentace obsahuje i hook pro propojení se samotným Sails.js. Sice nenabízí podporu SQL i NoSQL databází, neboť je stavěn pro databáze relační (tedy MySQL, PostgreSQL, MariaDB, SQLite a MSSQL [36]). To ovšem není v případě mé práce problém, neboť využívám jen PostgreSQL.

Po instalaci `sails-hook-sequelize` bylo potřeba přepsat jak samotné modely, tak i jejich volání v controllerech, nicméně se tato práce vyplatila.

4.2.1 Autentizace

Autentizace je proces, při kterém se ověřuje, zda uživatel je opravdu ten, za koho se vydává. Vzhledem k tomu, že aplikace je stavěna jako tzv. SPA, využívá pro svůj běh také REST API a samotná autentizace uživatele se tak nemůže spoléhat na session. Místo toho musí umět pracovat s bezstavovou autentizací.

Uživatel je v systému autentizován pomocí JSON web tokenu. Tento token je vygenerován vždy při přihlášení a je odeslán uživateli. Token je následně uložen do Local Storage prohlížeče, odkud je při každém požadavku vedoucím na server vybírán a přidáván do HTTP hlaviček jako Bearer token. Toto přidání tokenu do požadavku se děje za pomoci *interceptoru* v AngularJS pomocí následujícího kódu:

```
$httpProvider.interceptors.push (function ($q) {
  return {
    request: function (config) {
      var token = localStorage.getItem ('auth_token');
```

```
      if (token) {
        config.headers['Authorization'] = 'Bearer ' + token;
      }

      return config;
    }
  };
});
```

Při implementaci tohoto typu autentizace jsem vycházel z článku uveřejněného na adrese `https://thesabbir.com/how-to-use-json-web-token-authentication-with-sails-js/`.

4.2.2 Autorizace

Autorizace řeší otázku, zda *autentizovaný* uživatel má oprávnění vykonávat určitou akci. Autorizace probíhá na úrovni rolí uživatele v konkrétním prostředí a je řešena jak na straně frontend aplikace, tak samozřejmě i na straně API.

Na straně klientské aplikace jsou jednotlivé ovládací prvky pouze skryty. Uživatel s rolí *host* tak nevidí např. odkaz pro vstup do nastavení projektu, nebo mu jsou skryta tlačítka pro spuštění testů. Vzhledem k tomu, že aplikace je stavěna jako SPA, může si uživatel zobrazit zdrojový kód a skrytá tlačítka buď zobrazit, nebo na nich vyvolat kliknutí (např. pomocí konzole v prohlížeči) a tím pádem „obejít“ autorizaci. Z tohoto důvodu je nutné chránit i samotné API.

Metody na API jsou chráněny pomocí tzv. *service*, kterou jsem pro účely kontroly přístupových práv napsal. Pro ilustraci jejího použití přikládám metodu pro spuštění testu (resp. zařazení testu do fronty).

```
permissionChecker.canManage (req, res, {
  testsId: req.testId,
  roles: ['manager', 'tester']
}, function () {
  testRunner.addToQueue ({id: req.testId}, console.error);
  // do not wait until all data are stored in database
  return res.ok ();
});
```

Jak je vidět z uvedeného příkladu, *permissionChecker* (resp. její metoda *canManage*) přijímá čtyři parametry, kde první dva jsou objekty požadavku (req) a odpovědi (res) definované někde uvnitř frameworku Sails.js, třetí parametr je definice povolených rolí a identifikátor objektu, podle kterého lze

odvodit, zda má přihlášený uživatel pro požadovanou akci oprávnění či nikoliv. Čtvrtým parametrem je pak samotná funkce (předávaná jako callback), která se vykoná v případě, že uživatel je autorizován pro danou akci.

4.2.3 Proces testování

V předchozí části jsem uvedl ukázku kódu, který přidává nový test do fronty na otestování. V uvedeném příkladu měl tento kód demonstrovat způsob autorizace požadavků uživatele, nicméně na něj v této části naváži.

Pro tuto část aplikace jsou klíčové dva balíčky: `sails-hook-publisher` a `sails-hook-subscriber`. Pro označení celého testu jako úspěšný či neúspěšný se používá `sails-hook-cron`.

Po kliknutí na tlačítko pro spuštění testu se zavolá výše zmíněný kód. Vzhledem k tomu, že Javascript je jazyk asynchronní, výše uvedený kód nečeká na uložení posledního requestu do databáze a rovnou vrací odpověď uživateli. Samotné naplánování spuštění testu začíná voláním metody `addToQueue` ze service `testRunner`. Tato metoda vytvoří kopii testu a všech jeho requestů v databázi. Díky zkopírování dat se případná editace testu (případně i requestů) neprojeví v již vyhodnocených výsledcích. Po vložení každého requestu do databáze je vytvořena nová úloha pomocí `sails-hook-publisher`. Tuto úlohu (konkrétní jeden požadavek) zpracovává tzv. *worker*. Tento *worker* je zodpovědný za označení testu jako *evaluating* (tedy je označen za probíhající, ovšem pouze v případě, že test již není v tomto stavu). Dále zodpovídá za odeslání definovaného požadavku na požadovanou URL adresu a následné vyhodnocení odpovědi na základě definovaných validátorů.

Periodicky (jednou za 20 vteřin) je spouštěna úloha pro označení testů buď jako úspěšné (success), nebo neúspěšně (failed). Pro lepší ilustraci přikládám diagram stavů pro test, viz obrázek 4.1. Periodické spouštění je řešeno pomocí *hooku* `sails-hook-cron`.

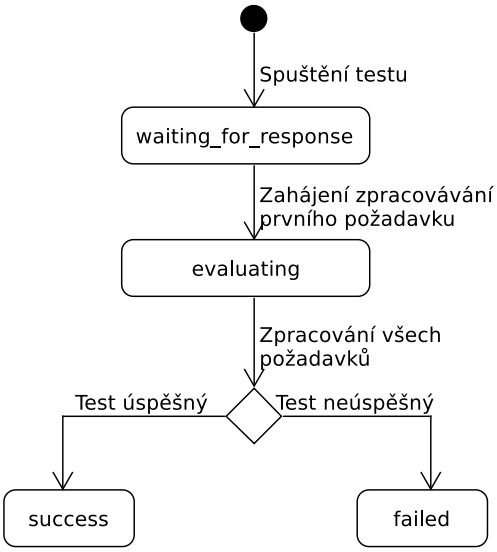
4.2.4 Přidání nového jazyka

Jedním z požadavků byla snadná správa překladů aplikace. Pro tento účel jsem využil rozšíření do AngularJS zvané `angular-translate`, s jehož pomocí je překlad aplikace poměrně jednoduchá záležitost.

Přeložení aplikace, resp. přidání nového jazyka, probíhá ve třech (čtyřech) krocích. Prvním krokem je úprava enumerace pro výběr jazyků aby bylo možné informaci o jazyku uložit. Přidání jazyka do definovaného typu v PostgreSQL databázi provedeme následujícím SQL příkazem:

```
ALTER TYPE languages ADD VALUE '<jazyk>';
```

Dalším krokem je samotný překlad aplikace. Pro tento účel je potřeba vytvořit nový soubor v adresáři `assets/locales` a pojmenovat ho podle kódu zvoleného jazyka `translation-<jazyk>.json`. Seznam všech překládaných



Obrázek 4.1: Stavový diagram pro test

řetězců se nachází v souboru `translation-cs.json`. Při tvobě překladů jsem použil formát překladů známý z nástroje Gettext, tedy že identifikátorem samotného překladu je text ve „výchozím“ jazyku. Samotné překládání pak spočívá v přeložení jen hodnoty atributů, tedy například takto:

```
{
...
"Jazyk": "Language",
"Jméno": "Name",
"Jméno parametru": "Parameter name",
...
}
```

Po přeložení tohoto katalogu zbývá úprava formuláře pro editaci uživatelského profilu. Aplikace bohužel nenačítá seznam dostupných jazyků z databáze, neboť jsem neočekával nějaké rapidní přidávání nových jazyků. Pro budoucí využití toto rozhodnutí pravděpodobně přehodnotím a udělám tento krok automaticky.

V tuto chvíli je potřeba upravit rozbalovací menu, ve kterém si uživatel může jazyk zvolit. Šablona pro uživatelský profil se nachází v adresáři `assets/templates` v souboru se jménem `user.html`. V kódu, který je dostupný v repozitáři je zakomentována volba pro angličtinu.

```
<select ng-model="user.profile.language">
  <option value="cs">Čeština</option>
  <option value="en">English</option>
```

```
</select>
```

Hodnota tagu `<option>` pro nově přidaný jazyk musí odpovídat zkratce jazyka, kterou jsem v předcházející části označoval jako `<jazyk>`.

V tuto chvíli je nový jazyk nainstalován a připraven k použití. Čtvrtým, nepovinným (ale vítaným) krokem je vytvoření pull-requestu v GitHubu, po jehož schválení bude nově vzniklý překlad dostupný v repozitáři i pro ostatní uživatele.

4.2.5 Úpravy ve wireframech

Mimo úpravy, které vyplynuly ze samotné heuristické analýzy, jsem učinil i jisté změny dle vlastního uvážení. Tyto změny vyplynuly jednak ze způsobu používání aplikace, jednak z omezení výpočetního výkonu a jednak ze samotných požadavků na funkcionalitu systému.

Wireframy nepočítaly s operátorem na porovnání získané a očekávané hodnoty. Proto jsem do okna s editací/tvorbou nového validátoru přidal rozbalovací menu s výběrem operátoru. V přehledu přiřazených validátorů se tato hodnota samozřejmě objevuje také.

Další věc, kterou jsem se rozhodl změnit, byl seznam testů v levém panelu. Původní návrh počítal s tím, že se bude zobrazovat jen pro aktuálně zvolené prostředí. To mi ovšem během vývoje přišlo ne zrovna šikovné, neboť by se tak lehce mohlo stát, že při práci v jenom prostředí by selhal test např. v jiném projektu a uživatel by se o problému vůbec nedozvěděl. Funkční prototyp tedy zobrazuje všechny vykonané testy ze všech prostředí, které uživatel aktuálně spravuje.

Další věcí, kterou jsem změnil oproti wireframům je zobrazení stavu prostředí. Původní návrh počítal s tím, že i v přehledu projektů budou graficky odlišeny prostředí, ve kterých poslední běh testů dopadl dobře, případně selhal. Od této myšlenky jsem však ustoupil, neboť se mi nepodařilo dostatečně spolehlivě a co možná s nejmenší prodlevou pro uživatele (a za využití co nejmenšího množství výpočetního výkonu) určit, který test se má počítat do posledního spuštění. Proto jsem se rozhodl graficky odlišovat jen úspěšné a neúspěšné testy.

4.2.6 Seznam použitých knihoven – backend

Pro vývoj aplikace jsem použil několik již existujících knihoven a balíčků. Všechny závislosti jsou instalovány pomocí balíčkovacího nástroje pro Node.js zvaný `npm` jejich aktualizace je tedy velmi snadná. Velká část z použitých knihoven je instalována jako závislost samotného frameworku Sails.js, z tohoto důvodu uvádím jen seznam ručně doinstalovaných balíčků. Kompletní seznam závislostí včetně aktuálně nainstalovaných verzí lze najít v souboru `package.json` v kořenovém adresáři projektu.

bcryptjs

Tato knihovna implementuje bcrypt do Node.js. V mé práci je použita pro hashování hesla.

- URL: `https://www.npmjs.com/package/bcryptjs`
- Licence: New-BSD / MIT

connect-redis

Tato knihovna zajišťuje spojení s Redis databází pro účely správy sessions. Aplikace sice sessions vyloženě nevyžaduje, ale instalací tohoto balíčku jsem odstranil jedno varování při spuštění aplikace v produkčním režimu. Redis je pro běh aplikace vyžadován i v jiném případě, než kvůli session.

- URL: `https://www.npmjs.com/package/connect-redis`
- Licence: MIT

crypto

Tento balíček je v aplikaci použit pro generování náhodných řetězců – jako nové, případně zapomenuté heslo. I přes fakt, že se jedná o velmi starý balíček, rozhodl jsem se jej využít kvůli jeho malé velikosti a nutnosti nulové konfigurace. Stránku na npmjs.com nemá vůbec použitelnou, uvádím tedy odkaz na GitHub.

- URL: `https://github.com/Gozala/crypto`
- Licence: dle informace v `package.json` BSD

jsonwebtoken

Knihovna zajišťující autentifikaci uživatele pomocí JSON web tokenu.

- URL: `https://www.npmjs.com/package/jsonwebtoken`
- Licence: MIT

nodemailer

Knihovna zajišťující odesílání e-mailů.

- URL: `https://www.npmjs.com/package/nodemailer`
- Licence: MIT

nodemailer-smtp-transport

Transporter umožňující knihovně `nodemailer` rozesílat e-mail pomocí SMTP protokolu.

- URL: `https://www.npmjs.com/package/nodemailer-smtp-transport`
- Licence: MIT

pg

Klient pro PostgreSQL databázi.

- URL: `https://www.npmjs.com/package/pg`
- Licence: MIT

request

Knihovna použitá pro odesílání HTTP požadavků. Tato knihovna tvoří základ pro samotné testování API.

- URL: `https://www.npmjs.com/package/request`
- Licence: Apache 2.0

sails-hook-cron

Tato knihovna umožňuje periodické spouštění úloh na pozadí v daný časový interval. Aplikace tuto knihovnu používá pro označování testů jako hotových a pro (nejen periodické) spouštění naplánovaných testů. Knihovna umožňuje plánování spuštění po vteřinách. A jak název napovídá, jedná se přímo o rozšíření samotného frameworku Sails.js.

- URL: `https://www.npmjs.com/package/sails-hook-cron`
- Licence: MIT

sails-hook-email

Tato knihovna poskytuje jednoduché propojení knihovny `nodemailer` a frameworku Sails.js.

- URL: `https://www.npmjs.com/package/sails-hook-email`
- Licence: dle informace v `package.json` MIT

sails-hook-publisher

Tato knihovna slouží jako producent pro úlohy na pozadí. Jedná se o nadstavbu nad knihovnu Kue a její integraci do Sails.js prostředí. V aplikaci se využívá pro vytvoření úlohy na vyhodnocení jednoho konkrétního requestu při testování. Tyto úlohy se ukládají do Redis úložiště.

- URL: `https://www.npmjs.com/package/sails-hook-publisher`
- Licence: MIT

sails-hook-sequelize

Propojení Sails.js frameworku s ORM Sequelize.

- URL: `https://www.npmjs.com/package/sails-hook-sequelize`
- Licence: MIT

sails-hook-subscriber

Tento hook do Sails.js je konzumentem úloh vytvořených pomocí doplňku `sails-hook-publisher`. V mojí aplikaci je použit právě pro vyhodnocení konkrétního jednoho requestu, který byl do Redis-u vložen jako úloha na pozadí. Stejně jako zmiňovaný producent pracuje s Kue.

- URL: `https://www.npmjs.com/package/sails-hook-subscriber`
- Licence: MIT

sequelize

ORM zajišťující práci s relačními databázemi. Použit jako náhrada za vestavěný Waterline ORM.

- URL: `https://www.npmjs.com/package/sequelize`
- Licence: MIT

4.2.7 Seznam použitých knihoven – frontend

Stejně jako v případě backendu, i pro vývoj frontendu jsem použil několik externích knihoven. V seznamu záměrně neuvádím AngularJS a Foundation, neboť jsem se o nich zmiňoval již v předcházející části textu.

UI router

Router pro frontend aplikace postavené nad AngularJS.

- URL: `https://github.com/angular-ui/ui-router`
- Licence: MIT

angular-gravatar

Tento doplněk do AngularJS zajišťuje jednoduché načítání avatarů uživatelů pomocí služby Gravatar (`http://en.gravatar.com/`).

- URL: `https://github.com/wallin/angular-gravatar`
- Licence: MIT

angular-sanitize

Knihovna zajišťující escapování HTML tagů z textů vypisovaných uživateli.

- URL: `https://github.com/angular/bower-angular-sanitize`
- Licence: MIT

angular-translate

Angular-translate je knihovna zajišťující překlady textů, což AngularJS sám o sobě nedokáže.

- URL: `https://github.com/angular-translate/angular-translate`
- Licence: MIT

angular-translate-loader-static-files

Tento modul pro angular-translate zajišťuje asynchronní načítání jazykových balíčků. Díky němu je možné ukládat veškeré překlady zvlášť v souborech a při změně jazyka jen načíst potřebný katalog.

- URL: `http://bit.ly/1VXQc8E`
- Licence: MIT

ngAnimate

Součást AngularJS poskytující animace v uživatelském prostředí. Díky této knihovně je možné poměrně jednoduše implementovat např. plynulý přechod mezi jednotlivými obrazovkami.

- URL: `http://www.nganimate.org/`
- Licence: MIT

jQuery

Knihovna umožňující jednoduchou manipulaci s HTML dokumenty, obsluhu událostí, Ajax a mnoho dalších věcí. Je instalována jako závislost Foundation frameworku, v aplikaci samotné její využití není příliš velké.

- URL: `http://jquery.com/`
- Licence: MIT

Foundation Datepicker

Jedná se o malá doplněk, který slouží k zobrazení okna pro výběr data a času. Využívá se u plánování spuštění testů. Pro svůj běh vyžaduje jQuery.

- URL: `http://bit.ly/1TUHULg`
- Licence: Apache 2.0

4.2.7.1 Foundation Icon font

Pro zobrazování ikonek v aplikaci jsem se rozhodl použít jeden z „obrázkových“ fontů, konkrétně font od zvoleného HTML frameworku Foundation. Výhodou fontu oproti vkládání obrázků je jednak škálovatelnost ikon (jednotlivé znaky jsou vektory) a jednak jednoduchá práce s nimi.

- URL: `http://zurb.com/playground/foundation-icon-fonts-3`
- Licence: MIT

4.3 Dostupnost funkčního prototypu

Na přiložené CD jsem vložil celý Git repozitář s aplikací. Funkční prototyp je dostupný ve větvi `funkcni_prototyp`. Případně lze naklonovat repozitář ze služby GitHub pomocí následujícího příkazu:

```
git clone https://github.com/michalkvacek/rest-api-tester.git \
-b funkcni_prototyp
```

Instalace aplikace je popsána v části Instalace na straně 67.

4.4 Testování funkčního prototypu

Pro otestování funkčního prototypu jsem zvolil jinou trojici expertů, než tomu bylo v případě testování wireframů. Učinil jsem tak jednak z toho důvodu, aby v analýze prostředí nehrála roli již předchozí znalost prostředí a jednak i proto, abych získal pohled na věc i od lehce odlišné skupiny uživatelů. Pro účely otestování funkčního prototypu jsem oslovil své dva spolužáky, jejichž bakalářská práce též obsahovala práci s REST API. Marek Dostál REST API implementoval a Jaroslav Veselý na základě Markova API stavěl aplikaci pro koncového uživatele. Jako třetího experta jsem vybral svého kolegu, Petera Hrdlicu, jehož hlavní pracovní náplní je programování frontend aplikací.

V následujícím textu uvedu konkrétní připomínky dotazovaných expertů, které následně shrnu do jedné tabulky. Z heuristik uvedu vždy jen ty, ke kterým měli dotazovaní nějakou připomínku.

Marek Dostál

1. Ví uživatel vždycky, kde se nachází (navigace). Ví uživatel vždy, v jakém je aplikace stavu?

- Lomítko v navigaci naprosto splývá s pozadím, spíš to vypadá jako menu. Závažnost: vysoká
- Na přehled projektů se lze dostat pouze přes navigaci. Závažnost: Střední

4. Poskytuje každá akce zpětnou vazbu srozumitelnou pro uživatele?

- Při úspěšném testu všechno svítilo zeleně, ale očekával jsem jasnou a zřetelnou informaci, zda se test povedl nebo ne. Závažnost: střední

6. Jsou všechny objekty, akce a jiné prvky vidět, kdykoliv je jich potřeba, aniž by si je uživatel musel pamatovat?

- Čekal bych, že do nastavení se dostanu i přímo z projektu. A ono to jde pouze z hlavní stránky. Závažnost: střední
- Docela dlouho jsem hledal tlačítko pro vytváření nových testů. Závažnost: střední
- Není mi úplně jasné, proč se tlačítkem „Development“ dostanu na stejné místo jako přes „Přehled prostředí“ → „Všechny testy“. Chybí mi menu, kde bych měl všechny operace „pod jednou střechou“. Závažnost: nízká

7. Jsou všechny akce, uspořádání a význam konzistentní s očekáváním cílové skupiny (zvyky odjinud...)

- Při změně projektu v levém menu bych očekával přepnutí se do něj, a ona se změní jenom ta navigace nahoře. Závažnost: vysoká

9. Je plocha zobrazení využita přiměřeně (vzhledem k celkovému vzhledu)?

- Jeden projekt vypadá na úvodní stránce trochu opuštěně, ale jinak v pořádku. Závažnost: triviální

Peter Hrdlica

1. Ví uživatel vždycky, kde se nachází (navigace). Ví uživatel vždy, v jakém je aplikace stavu?

- Lave menu → projekt vs prostredie: u projektu pridat napr text projekt, graficky odlísit napr aspon vacsim pismom, pripadne inym ohraniceni. Závažnosť stredná

5. Může se uživatel dostat z každé situace? Obejde se takové opuštění bez nutnosti opakovat dlouhou sekvenci akcí? Je použití tlačítek zpět a vpřed správné?

- Z detailu uz prevedeneho testu nemozem ist do editacie testu aby som napr mohol zmenit validator toho testu ked som zadal zly, alebo omylom ziadny. Závažnosť vysoká

6. Jsou všechny objekty, akce a jiné prvky vidět, kdykoliv je jich potřeba, aniž by si je uživatel musel pamatovat?

- Občas zabudnem, že sa nový prvok pridáva cez tlačidlo vpravo dole (hlavne na veľkej zobrazovacej ploche). Navrhol by som toto tlačidlo adekvátne premiestniť viac na úroveň očí pri prezeraní obsahu. Závažnosť vysoká

7. Jsou všechny akce, uspořádání a význam konzistentní s očekáváním cílové skupiny (zvyky odjinud...)

- Prepnutie projektu v ľavom menu neprepne zobrazenie na daný projekt - závažnost vysoká

- Pri vytvorení nového prvku, ako napr nový test, by som očakával, že sa mi rovno otvorí jeho detail pre editáciu a možnosť pridávania requestov atp. Takto musím robiť o jeden krok naviac a zdá sa mi to nepohodlné, platí napr aj pre vytváranie projektu. Závažnosť stredná

8. Odpovídá vzhled aplikace a ovládacích prvků cílové skupině?

- Chvíľami je obsah stránky až príliš farebný a je to mätúce = neviem (opticky) kde mám hľadať tlačídlá, ktoré zrovna potrebujem pre svoj úkon keď ich je tam vidieť tak moc, Závažnosť stredná

Jaroslav Veselý

1. Ví uživatel vždycky, kde se nachází (navigace). Ví uživatel vždy, v jakém je aplikace stavu?

- Lomítko u navigace ruší nebo je málo výrazné (buď odstranit nebo zvýraznit, je to něco mezi). Závažnost: střední

6. Jsou všechny objekty, akce a jiné prvky vidět, kdykoliv je jich potřeba, aniž by si je uživatel musel pamatovat?

- očekával bych možnost úpravy projektu i v jiných částech aplikace, ne jenom při přehledu projektů. Závažnost: střední
- Velmi dlouho jsem nemohl najít tlačítko pro přidání nového projektu a testu. Zelené „+“ není na první pohled úplně viditelné. Závažnost: vysoká

7. Jsou všechny akce, uspořádání a význam konzistentní s očekáváním cílové skupiny (zvyky odjinud...)

- Levé menu je trochu matoucí, první položka přepíná menu a nikam neodkazuje. Závažnost: střední
- V prohlížečích se nepoužívá „+“ vpravo dole pro přidávání. Závažnost: nízká

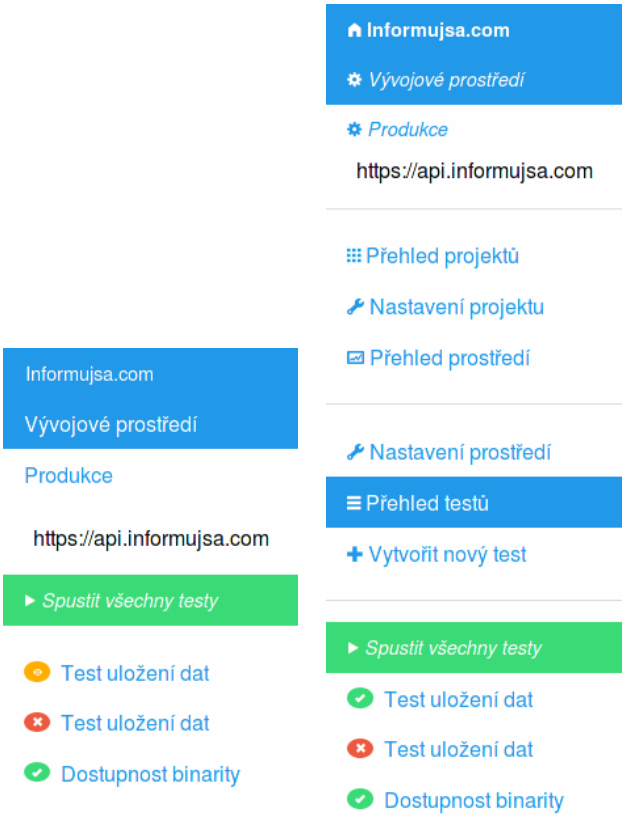
4.4.1 Vyhodnocení

Heuristická analýza provedená na funkčním prototypu odhalila další problémy v použitelnosti. Tyto problémy jsem se pokusil shrnout v tabulce 4.2. Závažnost uvedu vždy jako průměr ze závažností uvedených expertů. V případě, že by závažnost měla vyjít „nerozhodně“, bude v tabulce uvedena závažnost vyšší.

4.4.2 Závěr

Jak je vidět z tabulky 4.2, funkční prototyp obsahuje několik problémů, které opravím ve finální aplikaci. Osobně mne překvapilo, že experti při testování funkčního prototypu označili za problém chybějící menu, případně jednodušší přístup k nastavení prostředí a samotného projektu. Dle mého názoru v této věci hraje hlavní role samostatná práce s aplikací a zároveň samotná funkčnost. Na základě okamžité zpětné vazby tak vznikají nové požadavky a nové cesty při průchodu aplikací.

Na základě nasbíraných podnětů upravím uživatelské prostředí finální aplikace. V tuto chvíli je již samotný funkční prototyp plně fungující aplikace, tudíž se bude jednat jen o úpravy uživatelského prostředí.



(a) Sidebar uveřejněný v prototypu

(b) Sidebar ve finální aplikaci

Obrázek 4.2: Porovnání sidebaru ve funkčním prototypu a finální aplikaci

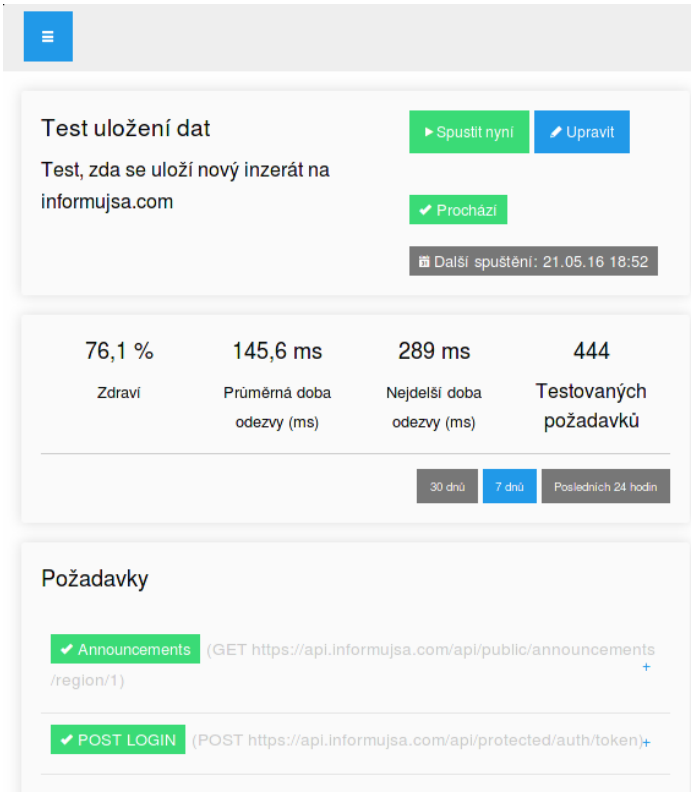
4.5 Implementace finální aplikace

Jak jsem uvedl v předchozí části, implementace finální aplikace zahrnuje jen úpravy uživatelského prostředí, neboť samotná funkcionalita byla implementována již pro účely funkčního prototypu.

Finální aplikace je dostupná v Git repozitáři ve větvi `master`, případně je nasazena na doméně `http://rest.kvacek.cz`. Pro ilustraci zde uvádím pár obrázků, na kterých jsem se snažil zachytit změny oproti funkčnímu prototypu.

Další změnou bylo přidání tlačítka pro vytvoření projektu a prostředí přímo pod jejich výpis, neboť experti identifikovali jako problém špatně viditelné zelené tlačítko. Tlačítko pro vytvoření nového testu jsem přesunul do menu, odkud je dostupné ze všech částí aplikace (resp. tehdy, kdy je zvolené nějaké prostředí a vytvoření testu tedy dává smysl).

Největší změnu doznal levý panel, kam přibylo jednak menu a jednak jsem odlišil typem písma jednotlivé projekty a prostředí. Pro ilustraci přikládám obrázek 4.2.



Obrázek 4.3: Ukázka zobrazení prostředí na menším rozlišení (570 px na šířku)

Přílišnou barevnost tlačítek jsem vyřešil tak, že oranžová tlačítka jsem přebarvil na šedou, kulaté tlačítko pro přidání nového projektu, prostředí a testu jsem zpřístupnil jen pro menší rozlišení.

Jednou z posledních úprav, které jsem v uživatelském prostředí v rámci implementace finální aplikace učinil, že zpřístupnění uživatelského prostředí i pro mobilní zařízení. Rozhraní sice není dokonale responzivní, ale je možné ho používat i v menších rozlišeních. Pro ilustraci přikládám screenshot 4.3.

4.6 Podněty k vylepšení

Přestože aplikace funguje, nelze říci, že veškeré její komponenty byly řešeny ideální cestou. Uvádím zde seznam nápadů k vylepšení funkcionality aplikace a tím i ke zlepšení uživatelské přívětivosti.

4.6.1 Nasazení weboscketu

Jako první podnět k vylepšení uvedu zavedení websocketu pro získávání výsledků testu pro účely přehledu v levém menu. V tuto chvíli aplikace odesílá

každých 30 vteřin dotaz na server s žádostí o všechny výsledky testů za poslední 2 hodiny. Samotné výsledky se pak vybírají na základě dat přijatých z předchozího požadavku. Nasazením websocketů by se snížilo nejen vytížení serveru, ale i množství dat odesílaných (a přijímaných po síti) a zároveň by se snížila doba mezi reálným vyhodnocením testu a doručením této informace uživateli. Výsledky by se totiž odesílaly ihned po změně, ne jen jednou za 30 vteřin.

Framework Sails.JS v sobě implementuje knihovnu Socket.io (`http://socket.io/`). Samotnou knihovnu obaluje do sady vlastních funkcí, které jsou dostupné pod názvem sails.io.js [37]. Dále existuje doplněk pro Sequelize, který nabízí podporu Publish/Subscribe [38]. Toto rozšíření lze najít v repozitáři `npm` pod jménem `sails-hook-sequelize-pubsub`. Bohužel se mi ale nepodařilo tyto dvě služby zprovoznit, z tohoto důvodu aplikace využívá tzv. *Ajax polling*.

4.6.2 Databáze

Tuto část jsem již zmínil v kapitole týkající se volby technologií. V tuto chvíli veškerá aplikace využívá jednu jedinou databázi, kterou je PostgreSQL. Pro stávající potřeby dostačuje, nicméně by stálo za úvahu databázi rozdělit na SQL a NoSQL část. V NoSQL části by se vyskytovala veškerá data, která se přímo týkají výsledků testů. Ve stávající implementaci tedy konkrétně tabulky `runnedTests`, `responses` a `evaluatedAssertions`.

U výsledků testů lze očekávat vysokou náročnost na velikost úložiště (neboť se ukládá i kompletní odpověď ze strany testovaného API). V Node.js lze využít MongoDB, případně lze zvolit databázi Cassandra, která je na zápis optimalizována [39]. Do prostředí Node.js lze integrovat pomocí ovladače napsaného přímo výrobcem – firmou Datastax. Tento driver je dostupný v npm repozitáři jako balíček `cassandra-driver`.

4.6.3 Integrace testovacích nástrojů třetích stran

Díky open-source licenci je možné zdarma využívat služby integrované do služby GitHub. Jako příklad uvedu nástroj na continuous integration Travis CI (`https://travis-ci.com/`), nebo CircleCI (`https://circleci.com/`). Pro kompletní seznam služeb integrovaných do repozitáře na GitHubu doporučuji nahlédnout na celkový přehled dostupný na adrese `https://github.com/integrations`.

4.6.4 Design aplikace

V tuto chvíli nelze příliš mluvit o designu aplikace. Celkový vzhled se velmi opírá o navržené wireframy a to včetně barevného pojetí. Oproti návrhu jsem jak do funkčního prototypu tak do finální aplikace použil světlejší odstíny šedivé, barevné pojetí tlačítek a komponent využívaných z frameworku Foundation však zůstalo nezměněno. Pro produkční nasazení mezi širší skupinu

uživatelů by určitě bylo vhodné zaměřit se více na barevné sladění detailů aplikace. V tuto chvíli může celkový vzhled působit lehce „šablonovitě“.

1. Aktuální stav aplikace	
Bez nalezených problémů	
2. Konzistence terminologie	
Bez nalezených problémů	
3. Srozumitelnost terminologie	
Request endpoint a API endpoint	triviální
4. Zpětná vazba	
Absence zobrazení chyby	znemožňuje p.
Informace o žádných datech	střední
Absence zpětné vazby při uložení	vysoká
5. Možnost návratu z každého stavu	
Bez nalezených problémů	
6. Viditelnost prvků	
Zobrazení celkového výsledku testu v detailu	střední
„Zdraví testu“ v přehledu všech testů	střední
Výsledek testu v sidebaru vedoucí na detail testu	vysoká
Možnost naplánování spuštění přes editační formulář	střední
Informace o následujícím spuštění do přehledu testů	nízká
7. Očekávaná funkčnost ovládacích prvků	
Bez nalezených problémů	
8. Vzhled aplikace	
Příliš velký avatar v sidebaru	nízká
Nekonzistentní layout aplikace	střední
9. Využití plochy	
Prázdné místo v posledních neúspěšných testech	triviální
Ostatní	
Chybějící popis formátu pro vložení vlastnosti objektu	znemožňuje p.

Tabulka 4.1: Přehled problémů odhalených při testování wireframu aplikace

	Závažnost	Počet
1. Aktuální stav aplikace		
Přehlednost drobečkové navigace	vysoká	2
Splývající projekt a prostředí v sidebaru	střední	1
2. Konzistence terminologie		
Bez nalezených problémů		
3. Srozumitelnost terminologie		
Bez nalezených problémů		
4. Zpětná vazba		
Chybějící notifikace po vyhodnocení testu	střední	1
5. Možnost návratu z každého stavu		
Editace dotazu ze stránky s výsledkem	vysoká	1
6. Viditelnost prvků		
Tlačítko pro vytvoření nového projektu/testu	vysoká	3
Možnost nastavení projektu/prostředí	vysoká	2
Menu	nízká	1
7. Očekávaná funkčnost ovládacích prvků		
Volba projektu při výběru v sidebaru	vysoká	3
Přesměrování po vytvoření projektu/testu	střední	1
Zelené „+“ pro přidávání	nízká	1
8. Vzhled aplikace		
Mnoho barev tlačítek	střední	1
9. Využití plochy		
Pouze jeden vytvořený projekt v přehledu	triviální	1

Tabulka 4.2: Přehled problémů odhalených při testování funkčního prototypu

Kapitola 5

Testování

Testování je důležitou součástí implementace jakékoliv aplikace.

5.1 End to end testování

End to end testování je proces, při kterém se ověřuje funkčnost aplikace z pohledu uživatele. Pro účely otestování své aplikace jsem se rozhodl využít nástroj Protractor (`http://www.protractortest.org`). Ten je navržen speciálně pro testování aplikací postavených za využití frameworku AngularJS a díky tomu celý proces testování dokáže značně zjednodušit.

5.1.1 Instalace Protractoru

Samotný testovací nástroj v aplikaci dostupný není, je nutné ho doinstalovat. Pro instalaci Protractoru včetně potřebných závislostí se nejvíce hodí opět známý balíčkovací nástroj `npm`.

```
# npm install -g protractor
```

Tento příkaz nainstaluje Protractor jako globální aplikaci. Důležité je spustit příkaz pod uživatelem s právem zápisu do adresáře se všemi NodeJS moduly, např. root. Po instalaci tohoto balíčku je ještě zapotřebí nainstalovat/zaktualizovat tzv. Selenium server. Protractor však poskytuje svojí vlastní nadstavbu – WebDriver, tudíž instalace/aktualizace je opět otázkou jednoho příkazu.

```
# webdriver-manager update
```

Tento příkaz nainstaluje (případně zaktualizuje) potřebné závislosti.

5.1.2 Spuštění testů

Potřebné závislosti by měly být nainstalovány. Pro běh testů je potřeba nainstalovaný Selenium server spustit. Pro tento účel slouží následující příkaz:

```
# webdriver-manager start
```

Tento příkaz zpřístupní URL `http://127.0.0.1:4444/wd/hub`, což je adresa Selenium serveru, kterou využijeme v následující části. Posledním krokem je konfigurace Protractoru, která je uložena v `tests/e2e/e2e-conf.js`.

```
exports.config = {
  seleniumAddress: 'http://127.0.0.1:4444/wd/hub',
  baseUrl: 'http://localhost:1337',
  specs: ['e2e/*.js'],
  params: {
    user: '<přihlašovací e-mail>',
    password: '<heslo>',
    testingProject: <ID testovaného projektu>,
    testingEnvironment: <ID testovaného prostředí>
  },
  onPrepare: function() {
    browser.driver.manage().window().setSize(1600, 1200);
  },
  capabilities: {
    browserName: 'firefox',
    binary: '/usr/bin/firefox'
  }
};
```

Vlastnost `seleniumAddress` je URL adresa Selenium serveru, kterou jsme získali při spuštění příkazu `webdriver-manager start`. Další důležitou URL adresou je baseUrl, která odkazuje na naši testovanou aplikaci. V případě, že aplikace běží např. na jiném portu, je nutné tuto hodnotu upravit.

V sekci `params` se nachází globální proměnné dostupné z každého testu. V tomto případě se jedná o přihlašovací e-mail a heslo testujícího uživatele (uživatele pod kterým bude aplikace testována) a ID testovacího prostředí a testovacího projektu. Sekce capabilities pak označuje, jaký prohlížeč bude pro vyhodnocení testů použit. Pro detailnější popis konfigurace čtenáře odkáži raději na dokumentaci, která je dostupná v instalačním adresáři Protractoru v podadresáři `docs`.

Sekce `onPrepare` se spustí na začátku testování a slouží k nastavení rozměrů okna prohlížeče.

V tuto chvíli je Protractor připraven k použití a lze přejít k samotnému spuštění testů:

```
$ protractor tests/e2e-confg.js
```

Po chvilce čekání se otevře zvolený prohlížeč a začne vykonávat scénáře popsané v adresáři `tests/e2e`. Po dokončení testů se opět sám zavře a do konzole vypíše buď seznam chyb, nebo informace o úspěchu, viz ukázka níže.

```
$ protractor tests/e2e-conf.js
[18:15:01] I/hosted - Using the selenium server at
http://127.0.0.1:4444/wd/hub
[18:15:01] I/launcher - Running 1 instances of WebDriver
Started
...................................


35 specs, 0 failures
Finished in 149.192 seconds
[18:17:33] I/launcher - 0 instance(s) of WebDriver still running
[18:17:33] I/launcher - firefox #01 passed
```

Jak je vidět z tohoto výstupu, testy obsahují celkem 35 uživatelských akcí, které testují všechny klíčové části aplikace.

5.2 Testování výkonu API

Na aplikaci byl kladen požadavek, aby uživatel získal data ze serveru do 1 s. Pro účely testování jsem napsal jednoduchý skript v jazyce `bash`, který využívá nástroj `ab`.

Pro otestování rychlosti získávání dat ze serveru je nutný autentizační token a nastavení ID testovaných zdrojů. Token lze získat pomocí následujícího příkazu:

```
$ curl -X POST --data "email=<email>&password=<heslo>" \
  http://rest.kvacek.cz/api/v1/login
{
  "user": {
    "id":1,
    "name":"Michal Kváček",
    "email":"michal@kvacek.cz",
    "language":"cs",
    "createdAt":"2016-05-09T09:22:47.368Z",
    "updatedAt":"2016-05-15T16:29:26.599Z"
  },
  "token":"eyJ0eXAiOiJKV1Qi...g2JhWssfSkBjM"
}
```

Potřebný údaj je uveden jako atribut `token`. Dále je nutné vyplnit ID testovaného requestu, projektu, prostředí a výsledku testu. Tyto hodnoty se konfigurují přímo v souboru `tests/api-performance.sh`.

Pro spuštění skriptu, který ověří čas odpovědi serveru pak stačí z kořene projektu spustit příkaz

```
$ ./tests/api-performance.sh <token>
```

Výstup ze skriptu pak vypadá následovně:

```
/api/v1/assertions/types
Time per request:       113.266 [ms] (mean)
Time per request:       37.755 [ms] (mean, across
   all concurrent requests)

...
```

Skript vypíše do konzole čas potřebný pro vykonání požadavku na uvedenou URL adresu. Z tohoto výstupu lze nahlédnout, že všechny dotazované metody vrací výsledky mnohem rychleji, než je uvedeno v požadavcích.

5.3 Doplnění

V části o nastavení Protractoru jsem se zmínil o nastavení velikosti okna prohlížeče. Při testování se mi občas stalo, že Protractor označil element za nedostupný, neboť byl překryt vrchním panelem. Nástroj bohužel tuto situaci nevyhodnotil stejně, jak by učinil uživatel (tedy že by stránku posunul tak, aby tlačítko bylo viditelné a klikatelné). Nastavení velikosti okna tento problém odstranilo.

Vím, že *end to end testy* nejsou dostatečná záruka funkčnosti celé aplikace a určitě tak nemohu mluvit o plně otestované aplikaci. Těmito testy jsem chtěl jen zaručit správné fungování klientské části aplikace, každopádně pro následující rozvoj a provoz aplikace je rozhodně potřeba pokrýt kód alespoň jednotkovými testy. Z časových důvodů jsem tak bohužel nestihl učinit a aplikace je otestována jen manuálně.

Kapitola 6

Nasazení

V rámci nasazení popíšu dvě věci – jednak samotnou instalaci aplikace a jednak její aktualizaci.

6.1 Požadavky

Pro běh systému je vyžadován následující software:

- NodeJS 4.4.4 – 5.11.0
- PostgreSQL 9.5.2
- Redis 2.8.4 – 3.2.0
- npm 2.15.1 – 3.8.9

Uvedené verze nejsou limitující a je velmi pravděpodobné, že aplikace bude fungovat i na jiných verzích. Starší verze softwaru jsou nainstalovány na serveru, na kterém je aplikace dostupná z internetu, novější verze softwaru byla použita pro vývoj aplikace.

V případě, že aplikace má běžet na serveru, je nutné mít zprovozněný a nakonfigurovaný ještě webserver. Pro účely vývoje webserver nutný není.

6.2 Instalace

Instalace samotná spočívá v naklonování Git repozitáře, inicializace databáze, nainstalování závislostí, úpravy konfiguračního souboru a spuštění samotné aplikace.

Repozitář buď naklonujeme pomocí následujícího příkazu, nebo ho stačí stáhnout z GitHubu jako zip archiv. Osobně však velmi doporučuji způsob pomocí Gitu, neboť se tím usnadní i následné aktualizace.

```
git clone https://github.com/michalkvacek/rest-api-tester
```

Tento příkaz vytvoří adresář `rest-api-tester` v aktuálním adresáři a stáhne do něj aktuální data.

Druhým krokem je inicializace databáze. Tento krok se skládá ze tří částí – první z nich je vytvoření samotné databáze, následně vytvoření databázové struktury a třetím krokem je naplnění daty. Pro inicializaci databáze jsou potřebné dva soubory – `schema.sql` a `seed.sql`. Oba dva soubory se nachází v adresáři `database` v nově vzniklém adresáři `rest-api-tester`. Následující příkazy spouštím z kořenového adresáře projektu, tedy po naklonování aplikace se přesunu do vzniklého adresáře a inicializaci provádím tam.

```
# vytvoreni databaze
$ sudo -u postgres psql
psql (9.5.2)
Pro získání nápovědy napište "help".

postgres=# create database rest_api_tester;
CREATE DATABASE
postgres=# \q

# vytvoreni struktury databaze
$ sudo -u postgres psql rest_api_tester < database/schema.sql
CREATE TYPE
ALTER TYPE
CREATE SEQUENCE
...
ALTER TABLE

# vlozeni dat do tabulky assertions
$ sudo -u postgres psql test < database/seed.sql
SET
...
```

Po doběhnutí druhého příkazu je databáze připravena.

V dalším kroku je potřeba nainstalovat závislosti. Tyto závislosti jsou spravovány pomocí balíčkovacího nástroje pro NodeJS – npm. Pro instalaci spustíme příkaz `npm install`. Pozor, je nutné tento příkaz spouštět z kořene projektu (tedy z adresáře `rest-api-tester`), neboť pro instalaci vyžaduje soubor `package.json`. Tento příkaz nainstaluje všechny potřebné knihovny a po jeho doběhnutí se můžeme posunout na předposlední krok.

Zbývá vytvořit konfigurační soubor pro aplikaci, ve kterém si nastavíme připojení k databázi, přihlašovací údaje k SMTP serveru pro odesílání e-mailů a podobné věci, které repozitář záměrně neobsahuje.

Konfigurační soubor vytvoříme v `config/local.js`.

```
var smtpTransport = require ('nodemailer-smtp-transport');

module.exports = {
  environment: process.env.NODE_ENV || 'development',

  jwToken: {
    secret: '<nějaký náhodný řetězec>',
    expiresIn: 60 * 60 * 24 * 7 // expirace tokenu v minutách
  },

  # nastaveni emailu kvuli odesilani posty
  email: {
    service: 'nodemailer-smtp-transport',
    transporter: smtpTransport ({
      host: 'mail.example.com',
      port: 25,
      auth: {
        user: '<uživatelské jméno>',
        pass: '<heslo>'
      }
    })
  },

  # definice pripojeni k databazi
  connections: {
    postgreSQL: {
      database: '<jméno databáze>',
      user: '<uživatel databáze>',
      password: '<heslo>',
      options: {
        host: 'localhost'
      }
    }
  }
};
```

Tento příklad uvádí nastavení emailu na SMTP server. V případě potřeby lze použít libovolný transportér pro `nodemailer`.

V tomto souboru lze předefinovat i nastavení uložená ve verzovaných souborech uvnitř adresáře config. V případě, že je potřeba předefinovat např. připojení k Redis databázi kvůli ukládání session, stačí do tohoto souboru vložit následující:

```
sessions: {
```

```
  host: 'localhost',
  port: <port>,
  ttl: <TTL>,
  db: 0,
  pass: <heslo>,
  prefix: 'sess:',
}
```

Toto nastavení bude použito místo obsahu souboru `config/sessions.js`.

Před samotným spuštěním je nutné se přesvědčit, že uživatel pod kterým aplikace běží, má právo zápisu do adresáře, ve kterém je nainstalována. Sails.js vytváří totiž adresář `.tmp`, do kterého si ukládá veškeré soubory dostupné z aplikace – tzv. assets (Javascript, CSS, obrázky).

Po uložení souboru lze aplikaci spustit pomocí příkazu `sails lift`. Tento příkaz spustí aplikaci ve vývojářském režimu a aplikace by měla být dostupná na adrese `http://localhost:1337`. Pro spuštění aplikace v produkčním režimu slouží přepínač `--prod`.

6.3 Aktualizace

Pro aktualizaci je nejlepší použít opět Git. V případě, že aplikace byla nainstalována jiným způsobem (stažením dat z repozitáře jako zip archiv a následným rozbalením), je nutné aktualizovat soubory ručně. Pro aktualizaci souborů pomocí Gitu použijeme příkaz `git pull`, který stáhne změněné soubory.

Je možné, že v této části příkaz selže s odkazem na konflikt souboru `views/layout.ejs`. Děje se tak proto, protože uvnitř tohoto souboru se vyskytují tři části, které spravuje `Grunt` (seznam JS skriptů, CSS stylů a šablon pro AngularJS). V případě, že tento soubor nebyl ručně upravován, stačí změny zahodit pomocí příkazu:

```
git checkout -- views/layout.ejs
```

a spustit aktualizaci souborů znovu. Po stažení souborů je opět nainstalovat/aktualizovat závislosti. Pro tento účel opět poslouží příkaz `npm install`.

Co se týče otázky změn struktury databáze, lze použít `sequelize-cli`. Ten je instalován společně se závislostmi projektu. Nástroj je nicméně ale potřeba inicializovat, v této otázce však raději odkáži čtenáře na oficiální dokumentaci na adrese `http://bit.ly/1TDTiIC`. Po nastavení tohoto nástroje můžeme spustit databázové migrace příkazem:

```
./node_modules/sequelize-cli/bin/sequelize db:migrate
```

Posledním krokem je pak spuštění aplikace (v případě že neběžela), nebo její restart. V případě, že aplikace již běží, změny se projeví až po jejím restartu.

6.4 Nasazení na rest.kvacek.cz

Pro demonstrační účely jsem aplikaci nasadil na server, ze kterého je dostupná na URL adrese `http://rest.kvacek.cz`.

Tento server je provozován na OS Ubuntu společně s Apache 2.4.7 jako webserver. Fungují na něm aplikace psané jak v NodeJS, tak i v PHP. Pro kontrolu aplikací v NodeJS je používán `supervisor` (`http://supervisord.org/`). Apache je nastavený jako reverzní proxy server.

Nový proces spravovaný pomocí nástroje `supervisor` jsem přidal jako nový soubor v /etc/supervisor/conf.d/kvacek-rest-tester.conf s následujícím obsahem:

```
[program:kvacek-rest-tester]
environment=NODE_ENV=production,PORT=4002
command=node /var/www/node/kvacek-rest-tester/app.js
user=www-kvacek
group=www-kvacek
autostart=true
autorestart=true
stdout_logfile=/var/log/supervisor/kvacek-rest-tester.log
```

Jak je vidět z konfiguračního souboru, aplikace na serveru běží na portu 4002, v produkčním režimu a je nastavená na automatické spuštění a automatický restart. Aby `supervisor` zaznamenal změny, je potřeba ho restartovat a následně proces spustit. Pro tento účel jsem použil následující příkazy:

```
# service supervisor restart
# supervisorctl start kvacek-rest-tester
```

Druhou změnou byla úprava konfigurace Apache. Do souboru s konfigurací pro doménu `kvacek.cz` jsem přidal sekci s definicí nového `VirtualHost`-u, která vypadá takto:

```
<VirtualHost *:80>
  ServerName rest.kvacek.cz
  ProxyRequests on
  Options -Indexes
  ProxyPass / http://localhost:4002/
</VirtualHost>
```

Poslední úpravou byla změna práv uživatele (v tomto případě `www-kvacek`) tak, aby měl právo zápisu do adresáře /var/www/node/kvacek-rest-tester. Následně jsem obnovil nastavení Apache příkazem `service apache2 reload` a aplikace byla nasazena.

6.4.1 Poznámka k nasazení

V tuto chvíli je aplikace dostupná pouze na nezabezpečeném protokolu `http`. Pro reálné nasazení toto řešení není z hlediska bezpečnosti vůbec dostačující, neboť veškerá komunikace je přenášena nešifrovaně. Vhodnějším (a pro produkční nasazení nutným) protokolem je využití `https`.

Dále by jistě bylo potřeba upravit nastavení databáze a optimalizovat její výkon. PostgreSQL nasazený na uvedeném serveru využívá výchozí konfigurace, která je dodávána společně s instalačním balíkem.

Závěr

V úvodu práce jsem provedl průzkum existujících služeb, které umožňují testování REST API. Byl jsem překvapen, když jsem našel jen dvě služby, jejichž primárním účelem je právě testování samotného API a jednu službu, která spouštění testů sice umožňuje, ale testování není jejím primárním účelem. Očekával jsem, že podobných služeb bude více.

V následujících částech jsem provedl průzkum ohledně funkčnosti a způsobu používání podobné aplikace, ze kterého vyplynulo, že uživatelé nechtějí nejprve definovat všechny metody a parametry API a až následně doplňovat vstupní data, ale více uvítají jednoduché vložení samotného požadavku. Tomuto požadavku jsem uzpůsobil i následující části práce, kterými byla technická analýza a samotná implementace.

Hlavní částí implementace bylo navržení vhodného uživatelského rozhraní a jeho následné otestování. Pro účely testování rozhraní jsem se rozhodl zvolit Nielsenovu heuristickou analýzu. Testoval jsem wireframy, na základě kterých jsem následně navrhl funkční prototyp aplikace. Tento prototyp jsem nechal ještě jednou otestovat a na základě připomínek od tří potencionálních uživatelů upravil uživatelské prostředí do podoby finální aplikace. V závěru implementace jsem zdrojový kód uvolnil prostřednictvím služby GitHub jako open source pod licencí MIT. Aplikace je dostupná na URL adrese `http://rest.kvacek.cz`.

Touto prací se mi podařilo splnit i vedlejší cíle, kterými bylo seznámení se jednak se samotným návrhem a testování uživatelského prostředí a jednak i seznámení s novými technologiemi. Zvlášť zajímavé pro mně bylo testování wireframů, které jsem testoval jako obsluhovanou maketu. Nikdy před tím jsem nic podobného nedělal, proto nedokáži posoudit, zda byl celý průběh testování veden správně a zda jsem z něj vytěžil maximum.

Výsledná aplikace se jistě nemůže svojí funkční výbavou rovnat ani jedné z analyzovaných služeb, což ovšem už od samotného počátku práce nebylo cílem. Může ale sloužit jako jednoduchý nástroj pro základní ověření funkcionality menší aplikace využívající REST API.

Literatura

[1] Rodriguez, A.: *RESTful Web services: The basics [online]*. IBM, 2015, [cit. 2015-12-09]. Dostupné z: `https://www.ibm.com/developerworks/webservices/library/ws-restful/`

[2] *Client URL Library [online]*. 2015, [cit. 2015-12-09]. Dostupné z: `http://curl.haxx.se/docs/`

[3] The PHP Group: *Client URL Library [online]*. 2015, [cit. 2015-12-09]. Dostupné z: `http://php.net/manual/en/book.curl.php`

[4] Jacobsen, K.; Oberhumer, M. F.; Pudeyev, O.: *PycURL – A Python Interface To The cURL library [online]*. 2015, [cit. 2015-12-09]. Dostupné z: `http://pycurl.sourceforge.net/`

[5] Luracast: *Restler API Explorer [online]*. 2015, [cit. 2015-12-09]. Dostupné z: `https://github.com/Luracast/Restler-API-Explorer`

[6] Nielsen, J.: How to Conduct a Heuristic Evaluation [online]. 1995, [cit. 2015-12-03]. Dostupné z: `https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/`

[7] Schmidt, J.: Analýza příkladů webových rozhraní – heuristické vyhodnocení [online]. 2012, [cit. 2016-04-15]. Dostupné z: `https://edux.fit.cvut.cz/courses/BI-TUR/tutorials/03/start`

[8] Apiary: Apiary Pricing [online]. 2016, [cit. 2016-01-03]. Dostupné z: `https://apiary.io/pricing`

[9] Aggarwal, D.: *Test Runner [online]*. 2015, [cit. 2016-05-09]. Dostupné z: `https://docs.optimizory.com/display/vrest/Test+Runner`

[10] Aggarwal, D.: *JIRA Integration [online]*. 2016, [cit. 2016-05-09]. Dostupné z: `https://docs.optimizory.com/display/vrest/JIRA+Integration`

[11] Aggarwal, D.: *Continuous Integration [online]*. 2015, [cit. 2016-05-09]. Dostupné z: https://docs.optimizory.com/display/vrest/Continuous+Integration

[12] Aggarwal, D.: *Jenkins Server [online]*. 2016, [cit. 2016-05-09]. Dostupné z: https://docs.optimizory.com/display/vrest/Jenkins+Server

[13] Aggarwal, D.: *NodeJS - Grunt [online]*. 2015, [cit. 2016-05-09]. Dostupné z: https://docs.optimizory.com/display/vrest/NodeJS+-+Grunt

[14] vREST: Pricing [online]. 2015, [cit. 2015-12-03]. Dostupné z: https://vrest.io/plans

[15] Inc., R.: Pricing [online]. 2016, [cit. 2015-12-03]. Dostupné z: https://www.runscope.com/pricing-and-plans

[16] Runscope Inc.: *API Monitoring and Testing: Integrating with Build, Continuous Integration, Deployment and Hosting Tools [online]*. 2015, [cit. 2015-12-09]. Dostupné z: https://www.runscope.com/docs/api-testing/integrations

[17] Runscope Inc.: *Integrating Runscope API Monitoring with Zapier [online]*. 2015, [cit. 2015-12-09]. Dostupné z: https://www.runscope.com/docs/api-testing/zapier

[18] SmartBear Software: *API Testing Features [online]*. 2015, [cit. 2015-12-09]. Dostupné z: http://www.soapui.org/about-soapui/features.html

[19] Software, S.: Team Testing Support [online]. 2015, [cit. 2015-12-09]. Dostupné z: https://www.soapui.org/soapui-projects/team-testing-support.html

[20] Runscope Inc.: *Integrating Runscope API Monitoring with Slack [online]*. 2015, [cit. 2015-12-09]. Dostupné z: https://www.runscope.com/docs/api-testing/slack

[21] Ruby: *About Ruby [online]*. 2015, [cit. 2016-05-09]. Dostupné z: https://www.ruby-lang.org/en/about/

[22] Cardell, C.: Interview with GitHub co-founder and CEO Chris Wanstrath [online]. 2012, [cit. 2015-12-03]. Dostupné z: http://doeswhat.com/2012/03/06/interview-with-chris-wanstrath-github/

[23] GitLab: *Requirements [online]*. 2015, [cit. 2016-05-09]. Dostupné z: http://docs.gitlab.com/ce/install/requirements.html

[24] Weksler, M.: Large Scale Payments Systems and Ruby on Rails [online]. 2015, [cit. 2015-12-03]. Dostupné z: http://nerds.airbnb.com/large-scale-payments-systems-ruby-rails

[25] Venners, B.: The Making of Python [online]. 2003, [cit. 2016-05-05]. Dostupné z: http://www.artima.com/intv/pythonP.html

[26] Foundation, D. S.: Why Django? [online]. 2015, [cit. 2015-11-03]. Dostupné z: https://www.djangoproject.com/start/overview/

[27] Alex: 14 Popular Sites Powered by Django Web Framework [online]. 2013, [cit. 2016-05-13]. Dostupné z: http://codecondo.com/popular-websites-django/

[28] Dahl, R.: Node.js, Evented I/O for V8 Javascript [online]. 2009, [cit. 2016-01-02]. Dostupné z: http://www.jsconf.eu/2009/speaker/speakers_selected.html

[29] Node.js: How Uber Uses Node.JS to Scale Their Business [online]. 2016, [cit. 2016-05-12]. Dostupné z: https://nodejs.org/static/documents/casestudies/Nodejs-at-Uber.pdf

[30] R, M.: Going node.js at Netflix [online]. 2016, [cit. 2016-05-12]. Dostupné z: https://www.dev-metal.com/going-node-js-netflix-slides-micah-r-netflix/

[31] Sails.js: *Features [online]*. 2016, [cit. 2016-05-10]. Dostupné z: http://sailsjs.org/features

[32] MongoDB: Who's using MongoDB? [online]. 2016, [cit. 2016-05-15]. Dostupné z: https://www.mongodb.com/community/deployments

[33] PostgreSQL: *Replication, Clustering, and Connection Pooling [online]*. 2015, [cit. 2016-05-09]. Dostupné z: https://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling

[34] Linster, M.: Postgres Outperforms MongoDB and Ushers in New Developer Reality [online]. 2014, [cit. 2015-12-03]. Dostupné z: http://www.enterprisedb.com/postgres-plus-edb-blog/marc-linster/postgres-outperforms-mongodb-and-ushers-new-developer-reality

[35] Dolgov, D.: Compare incomparable: PostgreSQL vs Mysql vs Mongodb [online]. 2015, [cit. 2015-12-03]. Dostupné z: http://erthalion.info/2015/12/29/json-benchmarks/

[36] Sequelize: *Docs [online]*. 2016, [cit. 2016-05-09]. Dostupné z: http://docs.sequelizejs.com/en/latest/

[37] Sails.js: *Socket Client [online]*. 2016, [cit. 2016-05-11]. Dostupné z: http://sailsjs.org/documentation/reference/web-sockets/socket-client

[38] Sails.js: *Resourceful PubSub [online]*. 2016, [cit. 2016-05-11]. Dostupné z: http://sailsjs.org/documentation/reference/web-sockets/resourceful-pub-sub

[39] Hobbs, T.: *Basic Rules of Cassandra Data Modeling [online]*. 2015, [cit. 2016-05-11]. Dostupné z: http://www.datastax.com/dev/blog/basic-rules-of-cassandra-data-modeling

Příloha **A**

Seznam používaných pojmů

Ajax polling Neustálé (periodické) dotazování serveru

Backend část aplikace starající se o vlastní funkční logiku aplikace

Dataset data organizovaná ve strojově čitelném formátu (CSV, XLS, ...) určená pro dávkové načtení dat

Deployment nasazení aplikace do provozu

Issue tracker nástroj na evidenci úkolů, chyb a požadavků

Mockovaná metoda Metoda, která se tváří jako funkční, nicméně se jedná jen o „atrapu“

Test case sada testů určená pro spuštění v jedné dávce

Wireframe Návrh rozložení prvků aplikace, „drátěný model“

Příloha B

Seznam použitých zkratek

API Aplication Programming Interface

CI Continuous Integration

CSRF Cross Site Request Forgery

CSS Cascading Style Sheets

CSV Comma-separated values

GUI Graphical user interface

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IDE Integrated Development Environment

IM Instant Messenger

JS Javascript

JSON JavaScript Object Notation

MVC Model-View-Controller

NoSQL Not only Structured Query Language

NPM Node.js Package Manager

REST Representational State Transfer

SPA Single Page Application

SQL Structured Query Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

VCS Version Control System

Příloha C

Obsah přiloženého CD

- `db-model.dbm` .. Model databáze ve formátu DBM pro aplikaci pgModeler
- `thesis.pdf` ..... text práce ve formátu PDF
- `wireframe` ..... adresář s wireframy aplikace ve formátu HTML
- `src`
  - `impl` ..... Git repozitář implementované aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX.