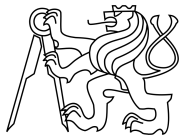
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	PerfRepo – nástroj pro reprezentaci výsledků výkonnostních testů a detekci výkonnostní regrese
Student:	Bc. Jiří Grunwald
Vedoucí:	Mgr. Martin Večeřa
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

PerfRepo je webová aplikace pro ukládání a analýzu výsledků výkonnostních testů a detekci výkonnostní regrese. Současné uživatelské rozhraní však již neodpovídá moderním trendům. Student má za úkol splnit následující body:
1. Popište nástroje pro výkonnostní testování a zaměřte se na nástroj PerfCake, pro který je tato aplikace primárně určena.
2. Proveďte analýzu současné verze aplikace, identifikujte její nedostatky ve funkčnosti i uživatelském rozhraní.
3. Proveďte analýzu moderních technologií pro tvorbu uživatelského rozhraní pro webové aplikace.
4. Navrhněte a implementujte nové uživatelské rozhraní na základě předchozích analýz nedostatků.
5. Navrhněte a implementujte webové služby a rozhraní pro servisní vrstvu.
6. Demonstrujte schopnosti nového uživatelského rozhraní na základě dat nahraných do aplikace skrze nástroj PerfCake.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 16. února 2017

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

Diplomová práce

PerfRepo – nástroj pro reprezentaci výsledků výkonnostních testů a detekci výkonnostní regrese

Bc. Jiří Grunwald

Vedoucí práce: Mgr. Martin Večeřa

4. května 2017

Poděkování

Chtěl bych velice poděkovat Martinu Večeřovi a Jiřímu Holušovi ze společnosti Red Hat za možnost podílet se na projektu PerfRepo. Děkuji jim za cenné rady a čas, který mi věnovali během psaní této práce. Dále bych chtěl poděkovat své rodině a přátelům za podporu.

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 4. května 2017 .....

Odkaz na tuto práci

Grunwald, Jiří. *PerfRepo – nástroj pro reprezentaci výsledků výkonnostních testů a detekci výkonnostní regrese.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato práce se zabývá přepracováním webové aplikace PerfRepo, která slouží jako úložiště a nástroj pro reprezentaci výsledků výkonnostních testů. Aplikace umožňuje porovnávat výsledky výkonnostních testů různých sestavení softwaru a tím odhalit případnou výkonnostní regresi. Cílem práce bylo odstranit nedostatky na funkčnosti současné verze a kompletně přepracovat uživatelské rozhraní, aby technologicky odpovídalo moderním trendům. To s sebou přineslo i přepracování webových služeb a dalších částí aplikace. Práce popisuje nástroje pro výkonnostní testování s ohledem na to, jakým způsobem je v nich realizováno vyhodnocení a ukládání výsledků. Aplikace je přidruženým projektem nástroje PerfCake (open source nástroj pro výkonnostní testování). Přínosem práce je zlepšení použitelnosti aplikace PerfRepo.

Klíčová slova výkonnostní testování, PerfCake, PerfRepo, úložiště výsledků výkonnostních testů

Abstract

This paper deals with the rework of the PerfRepo web application, which is used as a repository and a tool for representing performance test results. The application allows you to compare performance test results of different

builds of a software to detect possible performance regression. The aim of the thesis was to eliminate functionality deficiencies of the current version and to completely redesign the user interface of the application in order to technologically correspond to modern trends. It also includes the redesign of web services and other parts of the application. The paper describes the tools for performance testing with regard to how the evaluation and storage of the results is realized. The application is related project of PerfCake tool (open source performance testing tool). The benefit of this work is to improve the usability of PerfRepo application.

Keywords performance testing, PerfCake, PerfRepo, performance result repository

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Výkonnostní testování je jedním z prostředků pro zajištění a ověřování kvality software. Obecně se testováním zjišťuje rychlost, kapacita a stabilita systému. Cílem výkonnostního testování je vyhledat v systému slabá místa, která způsobují pokles výkonu či snižují stabilitu celého systému. Výkonnostní testování je velmi důležité u velkokapacitních systémů (servery, middleware, atd.), kde výkonové optimalizace můžou ušetřit nemalé zdroje.

Pro potřeby výkonnostního testování jsou k dispozici mnohé nástroje a frameworky. Úkolem těchto nástrojů je podrobit testovaný systém vhodně nasimulovanou zátěží a tento proces zaznamenat. Nejdůležitější je poté zpracovaní a reprezentace naměřených dat. To je možné provádět několika způsoby, od jednoduchého ukládání souborů s výsledky až po dedikovanou aplikaci s tvorbou dynamických reportů. Cílem je mít proces výkonnostního testování co nejvíce zautomatizován a provádět testování průběžně již během vývoje systému.

Jedním z testovacích frameworků je i nástroj PerfCake. Tato diplomová práce se zabývá aplikací PerfRepo, která je primárně vyvíjena jako úložiště a nástroj pro analýzu výsledků tohoto testovacího nástroje. Aplikace je ale navržena univerzálně a poskytuje webové API pro možnost interakce s jakýmkoliv systémem.

Kapitola 1

Cíl práce

Cílem práce je kompletně přepracovat uživatelské rozhraní webové aplikace PerfRepo, aby odpovídalo moderním trendům. Odstranění nedostatků současné verze a implementace nových požadavků si vyžádá změny v celé aplikaci, tedy od webových služeb, servisní vrstvy až po model aplikace. Cílem práce je tyto změny realizovat až po návrh rozhraní servisní vrstvy.

Jedním z úkolů je provést rešerši existujících nástrojů pro výkonnostní testování a jak se v nich řeší reprezentace výsledků. Podrobněji se má analýza věnovat nástroji PerfCake a jeho propojení s aplikací PerfRepo.

Část práce se má zabývat analýzou technologií pro tvorbu webových aplikací. Cílem je vybrat vhodné technologie pro aplikaci PerfRepo. Serverová část aplikace se nebude technologicky měnit a analýza se bude věnovat pouze technologiemi pro tvorbu frontend části webové aplikace.

Posledním úkolem je předvést možnosti nového uživatelského rozhraní. Od vytvoření definice testu, nahrání a zobrazení naměřených dat až po vytvoření a práci s reportem.

Kapitola 2

Popis problému

Výkonnostní testování je jedno z měřítek, kterým je možné stanovit kvalitu systému. Výkonnostní testování by se nemělo provádět pouze před vydáním nové verze systému, ale už během vývoje. Při průběžném testování je mnohem snadnější odhalit část zdrojového kódu, který způsobuje výkonnostní problémy. Z důvodu opakovaného spouštění těchto testů je nutné provádět tyto testy co nejvíce automatizovaně.

Obecně nelze otestovat všechny možné případy. Podle důležitosti systému a potřeb je nutné připravit vhodné testovací případy. Testovací prostředí by mělo být co nejpodobnější reálnému prostředí. Charakter a velikost zatížení by měl odpovídat reálnému použití, i malý nárůst zatížení může znamenat pád systému, a to jen kvůli malé části systému (slabina, úzké místo). Podle charakteru systému je potřeba testovat zátěž pro různý počet simulovaných uživatelů z různých geografických regionů.

2.1 Výkonnostní testování

Výkonnostním testováním se sledují různá měřítka, může jít o čas odezvy, latenci, propustnost a další. Dále lze měřit využití zdrojů (například využití operační paměti).

U definice simulované zátěže se často používá pojem virtuální uživatel, to představuje simulovaného uživatele, který posílá požadavek na systém. Simulovaných uživatelů může být desítky, tisíce, sta tisíce, ale i pouze jeden uživatel, záleží na testovaném systému. V případě většího počtu uživatelů, je často vhodné přidávat uživatele postupně (tzv. nastavení „ramp-up“), nepustit všechny jejich požadavky najednou. To jinak může způsobit umělé slabiny systému, ke kterým by reálně nedošlo (například na začátku dlouhého procesu je potřeba provést krátkou náročnou operaci, ke které později už nedochází).

Důležitou metodou je regresní testování, to se týká všech typů testů (nejen výkonnostních, ale i například funkčních). Regresní testování je opětovné otestování již existujících částí aplikace z toho důvodu, že po úpravě nové části aplikace mohla být narušena existující funkcionalita. Navíc i malá změna v programu může mít nečekaně velký dopad na výkon celého systému. Tyto testy je nutné provádět zcela automaticky pomocí CI[1], už jen z důvodu, že s vývojem aplikace stále roste počet testů a ruční spouštění a vyhodnocování by se časem stalo neúnosné[1].

2.1.1 Typy výkonnostních testů

Výkonnostní testy lze rozdělit do několika typů, mezi základní patří:

Load testy
: Obecný výkonnostní test, u kterého se nastaví požadovaná zátěž, která se opakováním testu nemění.

Soak testy
: Dlouho trvající testy běžící s nastavenou zátěží třeba celý den. Slouží pro odhalení chyb, které se mohou projevit až při delším spuštění systému (například odhalení memory leak).

Stress testy
: Systém je testován extrémní zátěží mimo běžné použití a testují se limity systému. Postupně se zvyšuje zátěž, dokud systém nespadne, hledají se tím úzká místa systému.

Scalability testy
: Test, při kterém se mění velikost zátěže a měří se dopad výkonu systému. Dále se těmito testy měří změna výkonu systému s měnícími se hardwarovými prostředky.

Benchmark testy
: Výkonnostní testy pro získání relativní hodnoty o výkonnosti systému. Testy slouží pro porovnání výkonu různých systémů řešící stejný problém. Případně pro porovnání výkonu různých verzí systému.

2.2 Ukládání a analýza výsledků výkonnostních testů

Naměřená data lze ukládat několika způsoby, podle toho se odvíjí i způsob následné analýzy výsledků. Výsledky lze reprezentovat z výstupu do konzole až po dedikovanou aplikaci, která výsledky dále interpretuje.

[1] Continuous Integration – průběžná integrace

K ukládaným výsledkům měření je nutné přiložit i množství metadat pro úspěšnou analýzu výsledků. Není na škodu ukládat i surové logy naměřených dat (pokud není problém s velikostí dat). Metadaty může být:

- verze systému (název, číslo ale i konkrétní commit hash z verzovacího systému),
- konfigurace spuštění (parametry spuštění testovaného systému),
- informace a konfigurace testovacího prostředí (např. hardwarové prostředky, název serveru),
- popis spuštění.

Jsou případy, kde je užitečné ukládat i surová vstupní data, se kterými bylo testováno. Lze poté opakovaně spustit identický test nebo spustit test v jiném prostředí. Problémem může být velikost dat.

Častým způsobem při vyhodnocování výsledků je zvolit si referenční hodnotu měřené veličiny (tzv. baseline). Tato hodnota slouží jako měřítko k naměřeným datům. Hodnota může například představovat mez akceptovaného výsledku nebo výsledek předchozí verze systému[2].

I během výkonnostního testování je nutné ověřovat správnou funkčnost a validovat příchozí odpovědi testovaného systému. Validace by ale měly být co nejjednodušší a jen formální, pro ověření správnosti jsou totiž určeny funkcionální testy. Například u webové služby často stačí testovat pouze status kód odpovědi, jelikož správnost úspěšné odpovědi systému předpokládáme.

2.2.1 Způsoby ukládání

Ukládání výsledků testů může být prováděno hned několika způsoby. Nejprimitivnějším způsobem je uložení výsledků do souboru. Souborem může být report ve formátu HTML či PDF s tabulkovou nebo grafovou reprezentací výsledku. Takto zpracovaný výstup je obtížné dále automaticky zpracovat pro další analýzu. Je proto vhodné uložit si výsledky i například do formátu CSV, XML pro pozdější zpracování dat. Tyto data lze poté zpracovat například v tabulkovém procesoru (Google Sheets, Microsoft Excel), nevýhodou je při běžném použití prakticky nulová automatizace[2].

Vhodným řešením pro dlouhodobou správu výsledků testů je použití dedikované aplikace. Výsledky jsou ukládané do databáze a teprve v aplikaci se

[2]V tabulkovém procesoru lze skriptovat a nahrávat data přes API, použití je ale náročnější

tvoří výsledné reporty. Výhodami je možnost vyhledávání v datech, tvorba dynamických reportů, sledování výkyvů (a upozornění na ně), správa uživatelů a jejich přístup k jednotlivým reportům. Příkladem může být aplikace PerfRepo.

Pro dlouho běžící testy je praktické výsledky ukládat do databáze typu time-series (TSDB). Tyto databáze pracují s časovou posloupností a příkladem takové databáze může být InfluxDB nebo Graphite. Výsledky lze pozorovat v reálném čase přímo z rozhraní databáze nebo použít jiný speciální nástroj pro analýzu těchto dat a přehledné zobrazení v grafech (Grafana, Influga)[3].

Další možností je reprezentovat výsledky výkonnostních testů v integračním nástroji pro automatizaci vývoje. Příkladem může být CI nástroj Jenkins rozšířený o Performance Plugin[4]. V aktuální verzi jsou již podporovány vstupní data ve formátu XML i CSV. Nevýhodou pluginu je, že nelze zobrazit průběžné výsledky a data jsou dostupná až po skončení testu[5].

Kapitola 3

Dostupná řešení

Kapitola popisuje nástroje pro výkonnostní testování. Popis se úplně nezabývá tím, jak dobře umí nástroje měřit výkon, ale jak se v nich řeší zpracování výsledků. Z vybraných nástrojů se kapitola zaměřuje na cílený PerfCake a nejznámější nástroj JMeter.

3.1 Apache JMeter

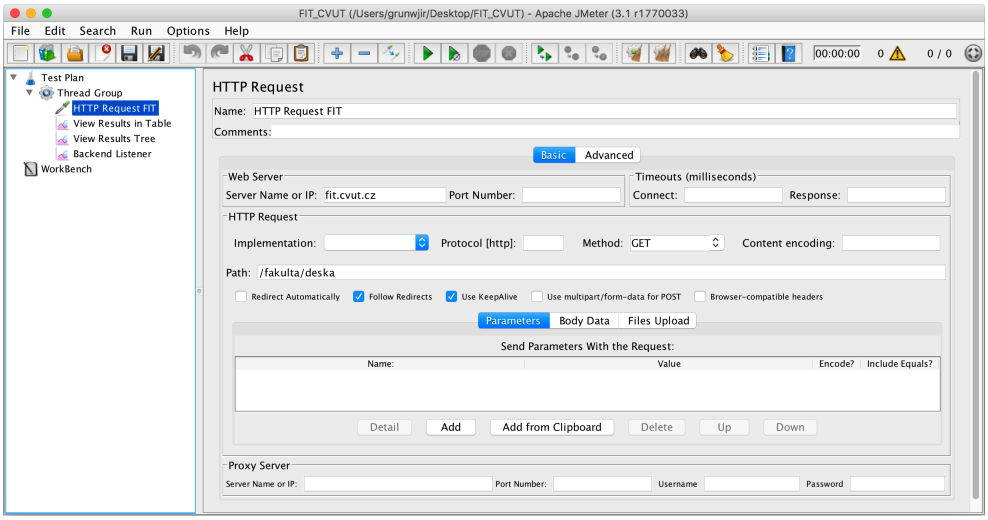
Zaběhlý nástroj pro testování zátěže napsaný v Javě (rok vydání 1998). Podporuje mnoho protokolů a rozhraní (např. HTTP, JDBC, LDAP, SOAP, JMS, FTP). Nástroj je rozšiřitelný pomocí pluginů a umožňuje velké množství konfigurace testovacího scénáře (řízení zátěže, navázání akcí před a po požadavku, čekání mezi požadavky). JMeter[6] pro každého simulovaného uživatele[3] vytváří jedno vlákno, tím pádem se testování tisíce uživatelů na jednom stroji stává nerealizovatelné (velká spotřeba prostředků na vytvořená vlákna). Nástroj ale podporuje distribuované testování a lze testovat z vícero strojů najednou (jeden funguje jako hlavní a shromažďuje výsledky měření).

Architekturou se jedná o desktop aplikaci s plnohodnotným GUI. Tvorba testovacích scénářů se provádí přímo přes grafické uživatelské rozhraní[4]. Konfigurace se generují do jednoho souboru, data jsou sice ve formátu XML, ale ruční konfigurace bez uživatelského rozhraní jsou v praxi neuskutečnitelné. Tento způsob řešení velmi snižuje znovupoužití konfigurací na jiných projektech.

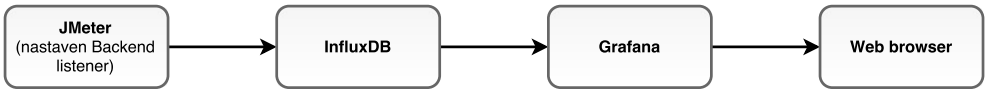
Výsledky měření jsou dostupné přímo v GUI aplikace, možné je vytvořit HTML report s grafy nebo uložit surové výsledky do souboru různých formátů (CSV, XML). Uložené výstupy jednotlivých testů do souboru je nutné

[3] Často se používá pojem virtuální klient

[4] Pro testování webových stránek existují nástroje, kterými lze nahrát jednotlivé kroky



Obrázek 3.1: Grafické rozhraní testovacího nástroje JMeter



Obrázek 3.2: Diagram integrace testovacího nástroje JMeter s nástrojem pro vizualizaci dat Grafana

dále zpracovat a hlavně zajistit jejich sdílené uložení. Porovnávání jednotlivých spuštění nebo zjištění výkonnostní regrese je nutné řešit dalšími nástroji, pluginy nebo službami. Dostupný plugin pro porovnání běhů je MergeResults[7], ale jeho možnosti a přizpůsobení jsou velmi omezené.

Další možností zpracování výsledků je za použití tzv. „Backend Listener“. Jde o externí programy, které implementují rozhraní[8] pro získání naměřených dat. Zpracování je poté libovolné. Příkladem může být implementace, která nahrává data do InfluxDB databáze, na kterou může být napojený nástroj Grafana, viz diagram 3.2. Tímto způsobem by mohlo být realizováno i propojení s aplikací PerfRepo.

3.1.1 BlazeMeter

Jednou z možností jak reprezentovat výsledky je použít službu BlazeMeter[9]. Její hlavní účel je spouštění JMeter testů, kde si můžeme nastavit počet simulovaných uživatelů a různé regiony světa, z kterých bude testováno. Služba

a provedené akce na webu. Výstupem je konfigurace testovaného objektu, dokončení konfigurace pak probíhá v nástroji JMeter

analyzuje data a podává reporty nad rámec možností nástroje JMeter. Například porovnávání jednotlivých běhů testů[10].

3.2 Gatling

Gatling[11] je open source nástroj vzniklý v roce 2011. Nástroj je napsaný v jazyce Scala, jedná se desktop aplikaci a zápis testovacích scénářů se provádí přímo ve zdrojovém kódu v jazyce Scala ve funkcionálním stylu. GUI pro tvorbu scénářů není dostupné. K dispozici je množství pluginů (např. Jenkins plugin, Maven plugin, Gradle plugin). Nástroj je primárně určený pro testování protokolů HTTP, JMS a WebSocket. Podpora distributivního testování je dostupná pouze v komerční verzi. Nástroj používá asynchronní model a pro více simulovaných uživatelů nevytváří separátní vlákno. Spouštění testů se provádí přes konzoli[12][13].

Výstupy jsou řešeny pomocí HTML reportu a nejsou upravitelné. Jedná se o report konkrétního spuštění. Pro analýzu výsledků mezi spuštěními je potřeba použít jiný nástroj, který zpracuje surová naměřená data.

3.3 Locust

Locust[14] je nástroj napsaný v jazyce Python pro výkonnostní testování nejen HTTP protokolu. Jedná se o multiplatformní nástroj s podporou distributivního testování. Konfigurace testovacího scénáře probíhá přímo zápisem kódu v jazyce Python, to zajišťuje snadné znovupoužití konfigurace v jiných projektech, možnost verzování konfigurací (Git, SVN). Také je velmi snadné přizpůsobení nástroje na požadovanou funkcionalitu. Nevýhodou je potřeba znát programovací jazyk Python, ale použité konstrukce nejsou těžké k naučení. Nástroj nepoužívá pro simulaci každého uživatele nové vlákno, lze tak simulovat velké množství uživatelů na jednom stroji[15].

Obsluha probíhá přes konzoly nebo webovou aplikaci, ve které jsou k nahlédnutí i výsledky testů. Ve webové aplikaci lze pozorovat i průběžné výsledky spuštěných testů (a test případně zrušit).

3.4 PerfCake

Open source nástroj pro výkonnostní testování napsaný v jazyce Java, který si klade za cíl podávat stabilní výsledky testování (za pomoci kvalitního generátoru zátěže) a co nejméně ovlivnit měřeným systémem testovací případ.

3.4.1 Popis a použití

Zápis scénářů se provádí v jazyce XML, DSL (Groovy syntaxe) nebo pomocí API (Java klient), všechny metody mají stejné možnosti. Nástroj nemá žádné GUI, ovládá se přes konzoli. Nástroj bohužel v současné verzi nepodporuje distributivní testování, ale je to jedna z budoucích plánovaných funkcionalit. Každý virtuální uživatel je zde reprezentovaný jedním vláknem, jedná se tedy o synchronní model.

Nástroj je vhodný pro testování vysokokapacitních systémů (middleware) a podporuje velkou škálu protokolů a rozhraní (např. HTTP, JMS, JDBC, SOAP, LDAP, MQTT). Velkou výhodou, co jiné nástroje běžně neumožňují, je použít různý protokol pro odeslání a přijmutí požadavku (například odeslat požadavek přes HTTP a přijmout odpověď přes JMS).

Možnosti použití jsou rozsáhlé, nástroj lze použít na zátěžové testy, stresové testy, soak testy či na testy škálovatelnosti.

3.4.2 Architektura nástroje

PerfCake má komponentový design. Aplikaci lze popsat a rozdělit do devíti komponent (schéma architektury viz obrázek 3.3):

Generator
: Definuje, jakým způsobem bude generována a odesílána zátěž na testovaný systém.

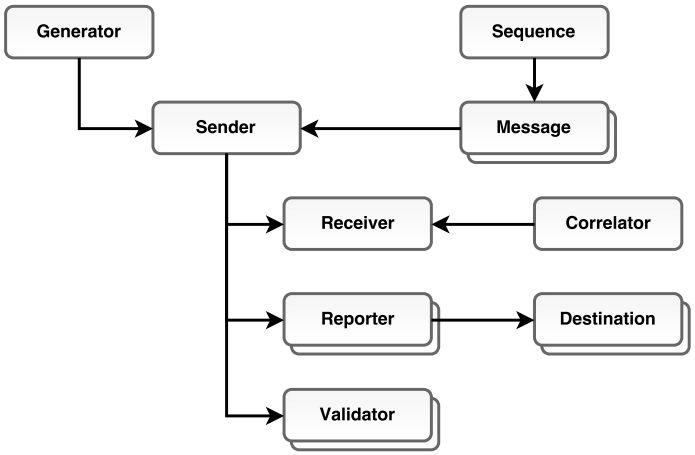
Message
: Specifikuje obsah, který je odesílán na testovaný systém. Každým požadavkem lze odeslat jinou zprávu.

Sequence
: Umožňuje dynamicky dodávat data při odesílání požadavku, obsah mu dodává komponenta `Message`.

Sender
: Specifikuje rozhraní testovaného systému zadáním jeho adresy a komunikačního protokolu.

Receiver
: Rozhraní pro příjem odpovědí z testovaného systému, protokol pro odeslání a příjem požadavků může být totiž různý.

Correlator
: Komponenta pro párování požadavků a odpovědí (nutné u rozdílných protokolů pro odeslání a příjem).



Obrázek 3.3: Architektura nástroje PerfCake (zdvojený rámeček ukazuje, kterých komponent může být více)

Reporter
: Určení jakým způsobem bude probíhat měření výsledků a co se měří (propustnost, počet iterací, využití paměti a další). Těchto reportérů může být v jednom testovacím scénáři více.

Destination
: Určení výstupu s naměřenými výsledky (konzole, HTML report, CSV, PerfRepo).

Validator
: Komponenta pro validaci odpovědí od testovaného systému.

3.4.3 Rozšíření funkcionality pomocí pluginů

Rozšíření funkcionality nástroje PerfCake je velmi snadné. Stačí pro požadovanou část z architektury (`sender`, `receiver`, `generator`, `reporter`, `destination` nebo `validator`) implementovat příslušné rozhraní. Tímto způsobem je řešeno i propojení s aplikací PerfRepo. Primární repozitář pro pluginy je na adrese `https://github.com/PerfCake/Plugins`, zde je dostupný i Maven archetyp pro snadné vytvoření nového pluginu. Vytvořenou zkompilovanou komponentu stačí nahrát do adresáře `$PERFCAKE_HOME/lib/plugins` a případné externí knihovny do adresáře `$PERFCAKE_HOME/lib/ext` (například JDBC ovladač konkrétní databáze)[16].

3.4.4 Projekty spojené s nástrojem PerfCake

S nástrojem PerfCake je přidruženo pár projektů, kde ne všechny jsou aktuálně v ideálním stavu. Existují pluginy do IDE Eclipse a Intellij IDEA, ale

bohužel jsou zastaralé a pro současnou verzi nepoužitelné. Plugin pro Intellij IDEA je ale pro aktuální verzi PerfCake v aktivním vývoji. Dalším projektem je vznikající kniha o výkonnostním testování v PerfCake. Plánem do budoucna je vytvoření grafického editoru pro tvorbu scénářů.

Kapitola 4

Současný stav aplikace

V současné době je aplikace PerfRepo produkčně používána na několika projektech[5], v poslední době ale docházelo pouze k malým úpravám a opravám chyb. Aplikace je používána například ve společnosti Red Hat.

Nedostatkem současné verze je nepřívětivé uživatelské rozhraní. Při prvním seznámení s aplikací je velmi obtížné s ní pracovat a někdy nezbývá jiná možnost než nahlédnout do její implementace. Aplikace má nedostatečné validace pro zadávání hodnot, a to poté způsobuje nepříjemné a neočekávané chyby.

Aplikace není dostupná jako služba a její vlastní spuštění je lehce náročnější kvůli nastavení aplikačního serveru a dodání připojení k relační databázi. To může uživatele odradit od vyzkoušení aplikace. Existuje již jedno řešení, a to je použití Dockeru, vytvořený image[6] se postará o veškeré nastavení (není momentálně aktuální s poslední verzí).

4.1 Popis aplikace

Aplikace nabízí API pro ukládání výsledků výkonnostních testů, naměřené hodnoty lze zadávat i přes uživatelské rozhraní. Jádrem aplikace jsou reporty, kterými lze porovnávat různé běhy testů a tvořit dynamické reporty pro sledování hodnot jednotlivých metrik.

Webová aplikace PerfRepo je napsaná na platformě Java EE a frontend část je vytvořená v technologii JSF 2.2. Pro běh aplikace se používá aplikační server WildFly. Ukládání dat probíhá do relační databáze PosgreSQL.

[5] Aplikace je používaná v určité míře na projektech Infinispan, Keycloak, OpenShift, Feedhenry a dalších

[6] Image je dostupný z `https://github.com/PerfCake/Docker.git`

4.2 Nedostatky na funkčnosti

Hlavním funkčním nedostatkem jsou nedostatečné validace při zadávání hodnot, špatně zadané údaje pak způsobují neočekávané problémy při další práci. Mezi další problémy patří:

- Prázdná domovská stránka aplikace, obsahující pouze informaci o aktuální verzi aplikace a odkaz na web pro hlášení chyb (vhodný by byl dashboard s užitečnými informacemi).
- Seznam vytvořených reportů nepodporuje stránkování a není žádná možnost filtrace podle názvu či typu reportu.
- Nelze vytvořit nového uživatele, je nutné ho vytvořit ručně v databázi.
- Nelze vytvořit novou skupinu a ani přiřazovat existující uživatele do existujících skupin.
- V detailu běhu testu nelze zjistit nesplněné podmínky pro naměřené hodnoty metriky. Pouze pokud má uživatel přihlášený odběr, tak mu přijde email.
- Porovnání běhů testů nelze uložit jako report, lze pouze vytvořit URL odkaz, ve kterém jsou obsaženy identifikátory jednotlivých běhů.

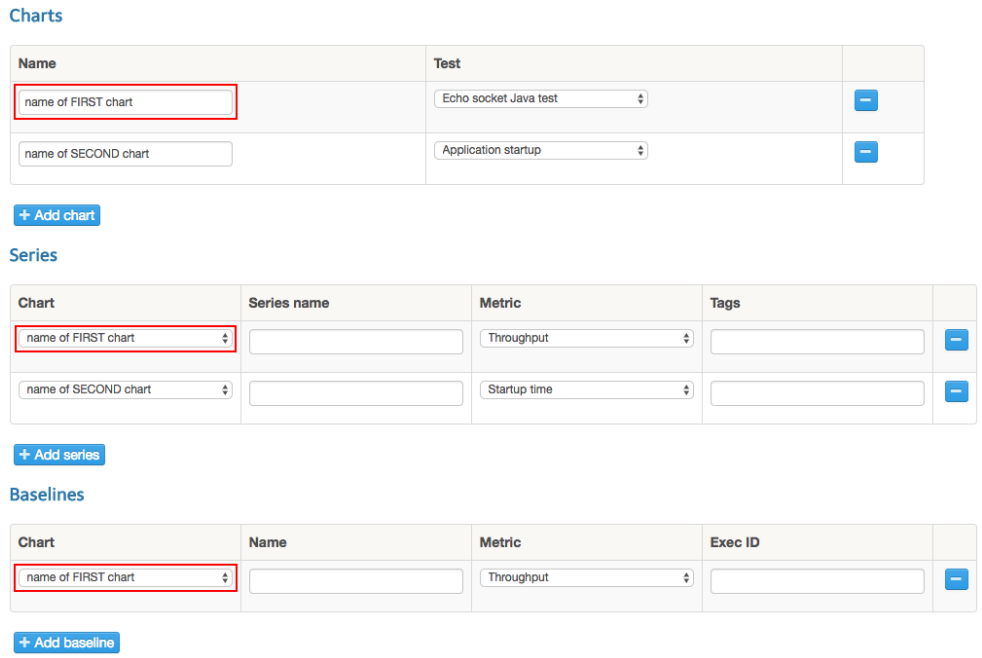
4.3 Nedostatky uživatelského rozhraní

Nedostatky na uživatelském rozhraní představují pro uživatele nekomfortní práci s aplikací, mezi nedostatky patří:

- Ve většině případů není podporováno našeptávání hodnot.
- Zadávání štítků je realizováno přes klasický textový vstup a jednotlivé štítky je nutné oddělovat mezerou. Uživatel nemá žádnou informaci o tom, jak štítky oddělit (například interpunkční čárka či středník neodděluje štítky).
- Zadávání hodnoty ze seznamu položek je realizováno pomocí výběrových polí[7], které v některých případech obsahují velké množství položek. Je to nepohodlné a taky náročné na zdroje (například výběr definice testu při konfiguraci reportu).
- Neuživatelské chybové hlášky (viz obr. 4.1).
- Nevhodné zobrazení vybraných běhů testů k porovnání. V menu je pouze zobrazena ikona nákupního košíku s počtem vybraných běhů.

[7] Výběrové pole – klasický `select` HTML element

Obrázek 4.1: Příklad chybové hlášky současné aplikace



Obrázek 4.2: Konfigurace reportu pro sledování hodnoty metriky v současné verzi. Na obrázku je vidět nevhodné zadávání sérií grafů. U série je nutné zvolit, ke kterému grafu patří.

- Nepřehledná konfigurace reportu, viz obr. 4.2, na kterém je konfigurace reportu pro sledování změn hodnot metrik. Série metrik a baseline (referenční hodnoty) se konfigurují na jednom místě pro všechny grafy. U každé položky je tedy nutné podle názvu grafu zvolit, ke kterému grafu patří. Pokud uživatel smaže název grafu (a opustí políčko), pak aplikace končí chybou a uživatel je přesměrován na hlavní stránku, tudíž přijde o veškerou rozpracovanou práci.

Webová aplikace nemá responzivní design, ale nejedná se o velký nedostatek s ohledem na určení nástroje. Optimalizace webu na malé obrazovky a mobilní telefony není prioritou.

Kapitola 5

Analýza a návrh

Kapitola se zabývá analýzou a návrhem nového řešení aplikace. Jsou popsány funkční požadavky a typické použití aplikace. Dále se kapitola zabývá návrhem uživatelského rozhraní nové verze a popisu technologií pro tvorbu webových aplikací (pouze technologie pro frontend).

5.1 Uživatelské skupiny aplikace

Aplikace PerfRepo má velmi úzce zaměřenou uživatelskou skupinu, aplikace je určena pro vývojáře a testery softwaru. Ti ji aktivně používají, další skupinou jsou QA[8] manažeři, kteří se většinou zajímají až o výsledné naměřené výsledky a reporty. Je proto nutné poskytnout možnost zveřejnit report i uživatelům, kteří s aplikací aktivně nepracují a nemají k ní plný přístup.

5.2 Sběr požadavků

Sběr požadavků probíhal s tvůrci a uživateli současné verze aplikace. Významným zdrojem požadavků byla webová stránka s nahlášenými chybami a požadavky[17], na které je umístěn nástroj JIRA[9]. Seznam požadavků byl vytvořen ve spolupráci s komunitou, která tento nástroj používá a vyvíjí.

5.3 Funkční požadavky

Seznam funkčních požadavků na aplikaci, které vyplynuly z analýzy. Požadavky se převážně týkají z pohledu uživatelského rozhraní.

[8] QA – Quality assurance (zajištění kvality)
[9] JIRA – nástroj pro evidenci chyb, problémů a podporu vývoje

5.3.1 Definice testu

Základním prvkem je definice testu, který reprezentuje testovací případ. Mezi požadavky patří tyto možnosti:

- Vytvořit, upravit, smazat a zobrazit definici testu.
- Zobrazit seznam všech testů s možností filtrování, řazení a podporou stránkování.
- U definice testu zadat název, unikátní textový identifikátor, skupinu, popis a sledované metriky. U metriky lze zadat název, popis a informaci, jestli vyšší hodnota metriky znamená lepší výsledek či naopak (neboli komparátor hodnoty pro porovnání).
- Definovat a upravovat podmínky kladené na naměřené hodnoty pro jednotlivé metriky testu. Kromě definice podmínky lze zadat název, štítky a popis.
- Umožnit uživateli přihlášení odběru nesplněných podmínek u proběhlých testů.
- Zobrazit seznam všech vykonaných testů pro vybraný test (rychlé vyfiltrování).

5.3.2 Běh testu

Běh testu (vykonání testu) reprezentuje instanci jednoho testu. Obsahuje informace o spuštění a samotná naměřená data. Mezi požadavky patří tyto možnosti:

- Vytvořit, upravit, smazat a zobrazit běh testu.
- Zobrazit seznam všech vykonaných testů s možností filtrování, řazení a stránkování obsahu.
- Provést hromadnou operaci nad vybranými položkami v seznamu vykonaných testů. Možností je:
  - přidat nebo odebrat štítky,
  - přidat nebo odebrat parametr o běhu,
  - odstranit běhy testu,
  - přidat běh testu do seznamu k porovnání.
- Zadat a upravit základní atributy běhu testu jako je název, čas spuštění, štítky a komentář.

- Zadat a upravit naměřenou hodnotu metriky. Naměřená metrika může obsahovat i sekvenci hodnot (umožnit zobrazení dat v grafu, včetně možnosti přepnout měřítko/parametr na hlavní ose).
- Přidat a upravovat parametry běhu, atributy jsou název a hodnota parametru. Umožnit vybrat parametr jako oblíbený. Hodnota parametru může být rozsáhlý text, ten je nutné v přehledu vhodně zobrazit (např. od určitého počtu znaků zobrazit odkaz, který zobrazí obsah v modálním okně).
- Stahovat a nahrávat souborové přílohy.
- Umožnit přidat běh testu do seznamu k porovnání.

5.3.3 Reporty

Hlavním prvkem aplikace jsou reporty, které vizualizují a pracují s naměřenými daty. Mezi obecné požadavky na každý typ reportu patří tyto možnosti:

- Vytvořit, upravit, smazat a zobrazit report.
- Zobrazit seznam všech reportů s možností filtrování, řazení a stránkování obsahu.
- Definovat přístupová práva na každý report. U položky lze definovat rozsah přístupu (na konkrétního uživatele, skupinu nebo jako veřejný report), typ přístupu (pouze čtení nebo i zápis). Veřejný report může být jen pro čtení.
- Označit report jako oblíbený, oblíbené reporty se uživateli zobrazí na dashboard stránce.

5.3.3.1 Report porovnávající hodnoty vykonaných testů

Tento report porovnává hodnoty explicitně vybraných běhů testů. Porovnávají se hodnoty běhů, pro které existuje společná metrika. Data se vizualizují v tabulce, která má vlastnosti:

- Sloupečky jsou jednotlivé běhy testů. Název sloupečku je zadaný alias běhu testu, název je odkazem na stránku s detailem běhu.
- Řádky jsou jednotlivé společné metriky s hodnotami.
- Buňka tabulky obsahuje procentuální změnu hodnoty od referenční hodnoty (baseline). Je nutné zobrazit i absolutní naměřenou hodnotu (např. po najetí na buňku nebo vedle procentuální změny).

- Buňka má nastavené pozadí na červenou, zelenou nebo neutrální barvu vzhledem k procentuální změně (např. pokud je změna větší než zvolený práh (threshold) a větší naměřená hodnota metriky znamená zlepšení výsledku, pak se buňka podbarví zeleně).
- Pro metriky, které obsahují sekvenci hodnot, bude buňka odkazovat na spojnicový graf porovnávající hodnoty.

Mezi požadavky na vytvoření reportu patří tyto možnosti:

- Definovat položky do porovnávací tabulky. Tabulky lze shlukovat do skupin. Pro skupinu lze definovat název a popis.
- Položkami jsou jednotlivé běhy, ty se definují pomocí:
  - názvu, který se objeví v hlavičce tabulky (alias, ale většinou se bude jednat o samotný název běhu testu),
  - možností, jak vybrat konkrétní běh (podle identifikátoru, filtru obsahující štítky nebo filtru obsahující parametry),
  - výběru jednoho běhu jako referenčního (baseline), od kterého se spočítají změny hodnot.
- Zvolit práh (threshold), podle kterého se budou zbarvovat buňky tabulky. Jednotkou jsou procenta.

5.3.3.2 Report pro průběžné sledování hodnoty metriky testu

Report slouží pro sledování změn hodnoty metriky v čase pro konkrétní definice testu. Report je určen pouze pro naměřené metriky s jednou hodnotou (pro sekvence je určený jiný druh reportu). Mezi požadavky patří tyto možnosti:

- Definovat graf se sledovanými metrikami, report může obsahovat více grafů. Pro každý graf umožnit zadat název a popis.
- Sledované metriky jsou reprezentovány jednotlivými sériemi grafu. Bod série v grafu je běh testu. Série se definují pomocí:
  - názvu, který se objeví v legendě grafu (většinou se bude jednat o samotný název metriky),
  - výběru testu (zdroj, ze kterého se vezmou vykonané testy),
  - výběru sledované metriky z testu,
  - běhy testu lze odfiltrovat pomocí zadání dotazu obsahující štítky nebo parametry.

Série bude ve výsledku zobrazovat posledních několik běhů testu, ale o počtu rozhodne servisní vrstva, která poskytne vypočítaná data k zobrazení.

- V grafu lze definovat speciální série určené pouze jedním během testu (tzv. baseline). V grafu půjde o horizontální přímku. Definují se pomocí:
  - názvu, který se objeví v legendě grafu,
  - zadáním konkrétního běhu testu,
  - výběr metriky (z dostupných ze zadaného běhu).
- Po kliknutí na bod v grafu zobrazit informace o běhu testu (název, asociovaná definice testu, oblíbené parametry spuštění) a umožnit přidat běh testu do seznamu k porovnání.

5.3.3.3 Report pro průběžné sledování metriky obsahující sekvenci hodnot

Report slouží pro sledování změn hodnoty metriky, která obsahuje sekvenci naměřených hodnot. Výsledek je reprezentovaný krabicovým grafem. Vzhledem k vlastnostem krabicového grafu bude možné v jednom grafu definovat pouze jednu sérii (při vícero sérií by se graf rozlézal moc do šířky). Mezi požadavky patří tyto možnosti:

- Definovat graf se sledovanou metrikou, report může obsahovat více grafů. Pro každý graf umožnit zadat název a popis.
- Sledovaná metrika je reprezentovaná krabicovým grafem. Jednou položkou grafu je běh testu. Série se definuje pomocí:
  - názvu, který se objeví v legendě grafu (většinou se bude jednat o samotný název metriky),
  - výběru testu (zdroj, ze kterého se vezmou vykonané testy),
  - výběru sledované metriky z testu,
  - běhy testu lze odfiltrovat pomocí zadání dotazu obsahující štítky nebo parametry.

  Série bude ve výsledku zobrazovat posledních několik běhů testu, o počtu rozhoduje servisní vrstva.
- Umožnit nastavit způsob pojmenování jednotlivých položek podle:
  - datumu spuštění testu,
  - parametru spuštění.
- Umožnit nastavit způsob řazení jednotlivých položek podle:

  - datumu spuštění testu,
  - parametru spuštění (řazení lexikograficky),
  - speciálního parametru spuštění (vyhledá v textu parametru verzi a řadí podle ní).

5.3.4 Rychlé porovnání vybraných běhů testů

V aplikaci bude možné vybírat jednotlivé běhy testů k rychlému porovnání. Aplikace bude udržovat seznam vybraných běhů testů k porovnání (vybrat běh testu bude možné z vyhledávání nebo například z reportu). Mezi požadavky patří tyto možnosti:

- Poskytnout přehledné zobrazení vybraných běhů testů.
- Manipulovat se seznamem vybraných běhů testů (odebírání položek).
- Zobrazit přehled porovnání běhů testů (reprezentace tabulkou).
- Umožnit vytvořit z přehledu plnohodnotný report (předvyplnění průvodce).

5.3.5 Dashboard stránka

Dashboard je stránka s přehledem aktuálních informací a usnadňuje přístup k důležitým funkcím systému. Aplikace bude nabízet jednoduchý přehled, mezi požadavky patří tyto možnosti:

- Zobrazit poslední proběhlé testy. Běh testu bude možné snadno přidat do seznamu k porovnání. Počet zobrazených běhů testů bude záviset na servisní vrstvě.
- Zobrazit seznam oblíbených reportů pro konkrétního uživatele.

5.3.6 Ostatní a obecné požadavky

Ostatní a obecné požadavky týkající se celé aplikace jsou:

- Možnost zadávat popisy a komentáře jako formátovaný text.
- Nastavení vyhledávání (filtry, řazení, počet záznamů na stránce a aktuální stránka) bude u definic testů, běhů testů a reportů perzistentní a bude ho poskytovat webová služba.
- Možnost změnit osobní údaje a heslo přihlášeného uživatele.

5.4 Scénáře užití

Scénáře užití popisují typické použití aplikace, kapitola obsahuje případy použití reportů.

5.4.1 Odhalení výkonnostní regrese

Scénář Projekt prochází živým vývojem a každý den vzniknou nové změny k zaintegrování. Je potřeba odhalit, jestli některá změna na programu nezpůsobí výkonnostní regresi (úbytek výkonu). Na projektu se používá průběžná integrace (CI[10]), která automaticky spouští výkonnostní testy a výsledky nahraje do aplikace PerfRepo. Kroky v aplikaci PerfRepo, které povedou k zajištění požadavku jsou:

1. Vytvořit definici testu a sledované metriky.
2. Nyní je nutné nastavit odesílání hodnot v testovacím nástroji na konkrétní definici testu (odesílání dat na REST API PerfRepo)[11].
3. Pro automatické upozornění uživatele na špatné výsledky testu je možné nadefinovat podmínky pro jednotlivé metriky. Podmínka se zapisuje v DSL a nabízí široké možnosti.
4. Vytvoření definice reportu pro sledování změn hodnoty metriky. Nastaví se zdrojový test a v něm sledované metriky. Report automaticky porovnává několik posledních běhů testů, to znamená, že report se mění s přibývajícími spuštěnými testy.
5. Výkonnostní regresi lze zjistit z nahlášeného upozornění nebo z výsledného grafu v reportu, z něhož lze vypozorovat i trend měnícího se výkonu.

5.4.2 Porovnání výkonů několika verzí systému

Vyvíjený projekt existuje v několika verzích s různou konfigurací, kde pro každou konfiguraci bylo provedeno výkonnostní testování. Nyní chceme tyto jednotlivé verze mezi sebou porovnat. Máme dvě možnosti k dosažení cíle. První možností jsou kroky:

1. Ve vyhledávání běhů testů vyfiltrovat požadované položky (filtrovat můžu například podle použitých štítků, nebo si vyfiltrovat běhy testu jen pro konkrétní definici testu).

[10] Continuous Integration – průběžná integrace

[11] Při použití testovacího nástroje PerfCake je situace jednodušší, v konfiguraci stačí přidat jako destinaci výsledků PerfRepo. K tomu slouží plugin pro propojení obou nástrojů, v nové verzi by měl plugin i vytvořit potřebné struktury v aplikaci a tím odpadne první krok

2. Vyfiltrované běhy testů si můžu přidat do seznamu k porovnání (buď jednotlivé položky, nebo celý zobrazený seznam).

3. Pro vybrané položky zobrazím porovnání společných hodnot (metrik). Reprezentace výsledku bude v tabulce.

4. Porovnání je teď možné uložit jako report, poupravit hodnoty a nastavit přístupová práva.

Druhou možností je vytvoření porovnání v průvodci pro vytvoření reportu, kde si vybereme jednotlivé běhy testu.

5.5 Technologie pro tvorbu webových aplikací

Kapitola se zabývá pouze technologiemi pro vytvoření frontend části webové aplikace. Serverová část bude pouze upravena na nové požadavky a nemá se technologicky měnit.

Komplexita webových aplikací je dnes stejná jako u klasických desktopových aplikací. Nároky také rostou se snahou zvyšovat uživatelský komfort při užívání aplikace. Vývoj komplexnější aplikace je již prakticky nutný si ulehčit použitím frameworku či nějaké knihovny. Technologie pokročily a existuje mnoho řešení. Pryč je doba knihovny jQuery, která sice ulehčovala manipulaci s DOMem, ale jen s pomocí ní napsat a udržovat komplexní webovou aplikaci bylo náročné.

5.5.1 Aplikační javascriptové frameworky

Kapitola se zabývá popisem nejznámějších javascriptových frameworků pro tvorbu webových aplikací. V popisovaných frameworcích lze vytvořit aplikace na principu SPA[12]. S těmito aplikacemi je zajištěn stejný uživatelský komfort jako s klasickými desktopovými aplikacemi. Stránky jsou dynamické a při změně obsahu nedochází k přenačítání celé stránky. Na pozadí se posílají požadavky na server a stahují se pouze potřebná data. K vyrenderování stránky (šablony s daty) pak dochází na straně klienta.

Mnohé frameworky přinášejí dvoucestný data-binding. Hodnotu zadanou z uživatelského rozhraní je potřeba projevit do modelu aplikace, a naopak změnou modelu je nutné aktualizovat webovou stránku. Frameworky to řeší automaticky za nás a velmi to ulehčuje vývoj a snižuje množství napsaného kódu. Způsob implementace dvoucestného data-bindingu se v jednotlivých frameworcích liší.

[12]SPA – Single page application

5.5.1.1 AngularJS

AngularJS je dnes nejpopulárnějším MVC frameworkem. Tento framework přinesl mnoho nových konceptů, které byly do té doby výhradou jen backendových technologií (dependency injection, speciální konstrukce pro servisní vrstvu, důraz na testovatelnost). Stavebním kamenem frameworku jsou direktivy (komponenty[13]), kterými lze tvořit nové HTML elementy a obohacovat chování jiných HTML elementů.

Tím nejhlavnějším je podpora dvoucestného data-bindingu, ten je implementován pomocí dirty checkingu a digest cyklu. To přináší u složitého uživatelského rozhraní s velkým počtem prvků výkonnostní problémy a obecně větší nároky na výpočetní prostředky. Není tedy moc vhodné použít AngularJS pro aplikace, které se budou užívat na mobilních zařízeních (kvůli baterii). Vhodným určením jsou interní informační systémy, kde většina uživatelského rozhraní jsou formuláře nebo statické zobrazení dat.

Framework nemá předepsanou strukturu aplikace. To přináší nekonzistenci mezi projekty s různými konvencemi. Během existence frameworku ale vzniklo několik doporučených konvencí a vzorů, nejznámější z nich je od autora John Papa[14].

Pro framework existuje velké množství rozšíření v podobě direktiv a komponent.

5.5.1.2 Angular 2

Angular 2 je kompletně nový framework, který není s první verzí kompatibilní. Nástupce Angular 2 řeší nedostatky první verze a v mnoha směrech je koncepčně úplně jiný. Framework je založený na tvorbě komponent, z kterých se poskládá celá stránka. Byl odebrán princip dirty checkingu[18] a zdroje uvádějí $5 - 10\times$ zrychlení[19] oproti první verzi. Nově framework přináší použití virtuálního DOMu a možnost vyrenderování šablony na serveru (mnohá inspirace od knihovny React), což urychluje prvotní načtení aplikace. Preferovaným jazykem je TypeScript, ale je možné použít i klasický Javascript, většina příkladů a dokumentace je ale vytvořena pro TypeScript.

Už použití TypeScriptu, který je nadstavbou Javascriptu a přináší například typovost proměnných, je Angular 2 zaměřen pro tvorbu větších (enterprise) aplikací.

[13] Od verze 1.5 bylo rozšířeno API a je možné vytvářet i komponenty

[14] Soupis vzorů a doporučení je dostupné na `https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md`. O popularitě návodu svědčí počet hvězdiček 22 232 (ohodnocení na serveru GitHub, údaj z 18. března 2017)

5.5.1.3 Ember.js

Ucelený javascriptový MVC framework pro tvorbu SPA aplikací. Podporuje dvoucestný data-binding a v současné verzi využívá virtuální DOM[20], který napomohl ke zvýšení výkonu[21]. Framework předepisuje pevně danou strukturu aplikace s návodem nejlepších postupů. Silnou stránkou je propracovaný CLI nástroj[22], který lze použít pro buildování aplikace a generování kódu pro typické konstrukce (například vytvoření souborů a kódu pro novou komponentu).

Dvoucestný data-binding je zde řešení pomocí tzv. accessorů. U objektů se k atributům nepřistupuje přímo, ale přes metody (getter, setter). Tím jsou zjištěny změny, které si přidávají do fronty změn. Tento přístup vyžaduje, aby objekty v modelu dědily od `Ember.Object`, nelze použít čistě POJO[15].

5.5.1.4 React

React je javascriptová knihovna pro tvorbu a vykreslování webových komponent. Nejedná se o MVC framework a jde o technologii, ve které si nadefinujeme zobrazení komponent podle příchozích dat, React poté efektivně manipuluje s DOMem[16] prohlížeče.

React tedy není uceleným frameworkem a vybrat kolekci technologií a zaintegrovat je mezi sebou nemusí být jednoduché, jsou ale k dispozici tzv. startovací stacky, které vybírají vhodné technologie za nás. React se nejčastěji používá s architekturou Flex.

5.5.2 Frameworky pro návrh uživatelského rozhraní

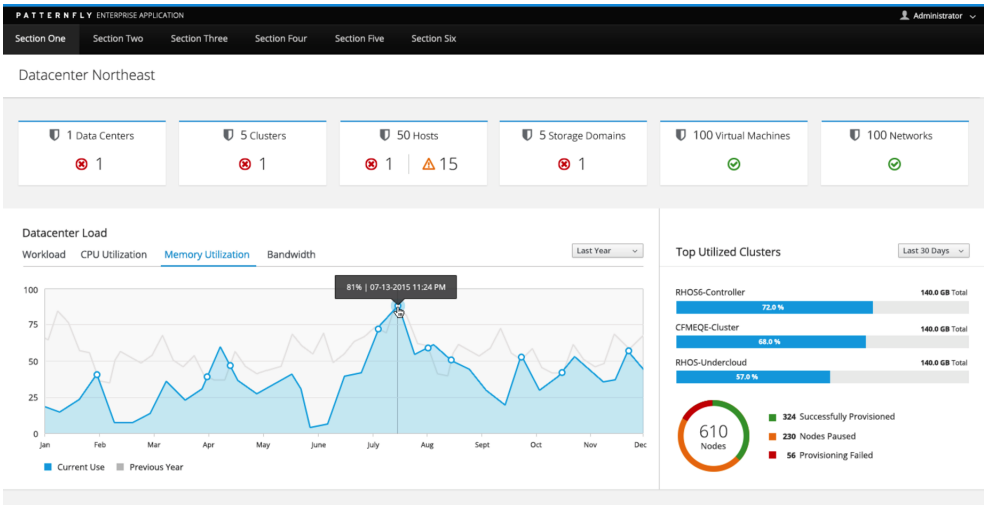
Někdy jsou tyto nástroje označovány také jako CSS frameworky. Přinášejí typografii a popisují vzhled a chování jednotlivých prvků webu, z kterých bychom měli být schopni poskládat celou naši aplikaci. Tyto frameworky většinou obsahují i referenční implementaci těchto komponent. Kapitola popisuje tři známé frameworky a jeden méně známý, který se specializuje na návrh enterprise aplikací.

5.5.2.1 Bootstrap

Nejznámější CSS framework s knihovnou komponent. Framework obsahuje responzivní grid systém, v mnoha projektech dojde k použití jen pouze této části frameworku. Velmi populární nástroj s masivní komunitou.

[15] POJO – Plain Old JavaScript Object

[16] Document Object Model – API pro manipulaci a přístup k obsahu v dokumentech HTML a XML



Obrázek 5.1: Ukázka návrhu dashboard stránky ve frameworku PatternFly

5.5.2.2 PatternFly

PatternFly[23] je UI framework specializující se na potřeby enterprise aplikací. Knihovna disponuje rozsáhlou dokumentací designu jednotlivých komponent a je k dispozici referenční implementace postavená nad frameworkem Bootstrap. Výhodou jsou návrhy komponent pro:

- rozmanité zobrazení obsahu (tabulka, seznam, karty),
- vyhledání, filtrování a řazení dat,
- zobrazení notifikací,
- rozvržení menu s akční lištou,
- vizualizaci dat v grafech,
- vytvoření rozsáhlého formuláře s více kroky pomocí průvodce,
- různé informativní panely vhodné pro dashboard stránku (ukázka obr. 5.1).

Sponzorem projektu je společnost Red Hat. PatternFly je použitý například pro webovou konzoli aplikace OpenShift[24].

5.5.2.3 Foundation

Foundation[25] je zaběhlý CSS framework s velmi kvalitním grid systémem pro tvorbu responzivních webů. První veřejné vydání proběhlo v roce 2011,

framework je vyvíjen společností ZURB pod licencí MIT. Oproti konkurenci nabízí extra podporu pro tvorbu responzivního obsahu HTML emailů. S nabídkou komponent je velmi podobný frameworku Bootstrap, jen s rozdílem designu.

5.5.2.4 Material Design

Material Design[26] představuje komplexního průvodce pro návrh uživatelského rozhraní napříč různými platformami. Rozsah použití je opravdu široký (webové aplikace, mobilní aplikace, rozhraní pro virtuální realitu, aplikace pro chytré hodinky a televize, rozhraní palubních systémů automobilů). Tvůrcem tohoto „vizuálního jazyka“ je společnost Google, která styl Material Design používá ve svém mobilním operačním systému Android, ale i v mnoha svých webových aplikacích (např. GMail, YouTube, Google Drive).

Na Material Design staví mnoho frameworků poskytující implementaci jednotlivých komponent. Oficiální implementací je Material Design Lite[27]. Oblíbenou implementací je poté framework Materialize[28], který se použitím velmi podobá frameworku Bootstrap[29].

5.6 Výběr technologií a návrh architektury aplikace

Klasická situace u webových stránek je, že se odešle požadavek na webový server, ten vrátí odpověď a webový prohlížeč s přenačtením stránky vykreslí výsledek. Lepším řešením bude vytvořit aplikaci typu SPA, která bude se serverem komunikovat na pozadí pomocí AJAX požadavků. Aplikaci bude vhodné rozdělit do několika vrstev a použít architekturu MVC.

Z výše zmiňovaných Javascriptových frameworků by nebyla chyba vybrat kterýkoliv z nich. Z CSS frameworků se nejlépe hodí PatternFly, který má nejlépe navržené komponenty pro zobrazení, filtrování a řazení dat. Výhodou frameworku je i referenční implementace postavená na frameworku Bootstrap 3. V podstatě se jedná o nadstavbu frameworku Bootstrap. PatternFly má také oficiální podporu pro AngularJS. S ohledem na výše zmíněné se použití frameworku AngularJS jeví jako dobrá volba.

5.7 Návrh uživatelského rozhraní

Návrh uživatelského rozhraní proběhl po prozkoumání možností frameworku PatternFly a výsledné návrhy wireframe jsou jím ovlivněny. Wireframe jsou nakresleny v online nástroji WireframePro od společnosti MockFlow[30].

5.7.1 Rozložení stránky

Prvním rozhodnutím bylo rozložení stránky, framework nabízí možnost jak vertikálního, tak horizontálního menu. Bylo vybráno vertikální menu, aby co nejvíce prostoru na výšku zůstalo pro obsah stránky. Navíc navrhované vertikální menu lze přepnutím zúžit a zobrazit pouze ikonky položek. Popis jednotlivých prvků pro wireframe 5.2:

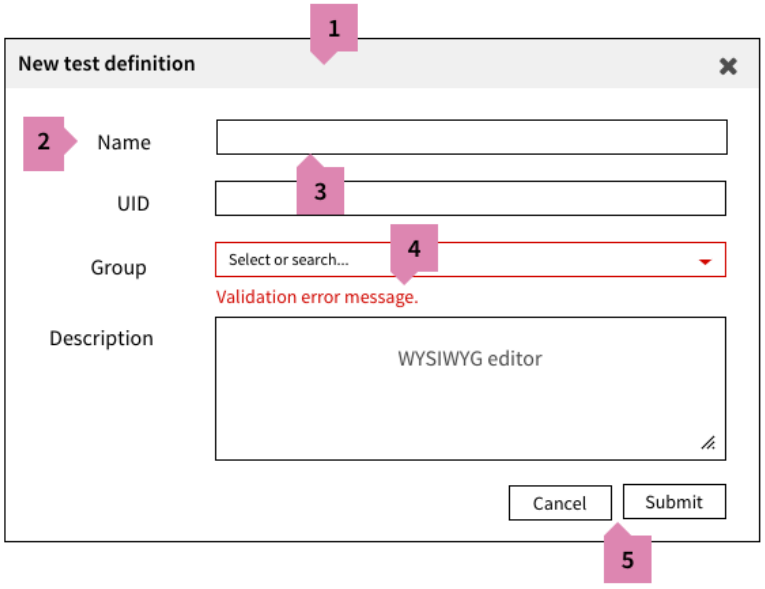
1. Vertikální menu, každá položka musí obsahovat ikonu. Ikony nejsou jenom estetickou záležitostí, menu lze zúžit a pak jsou zobrazeny pouze ikony. Aktivní položka menu bude zvýrazněna na levé straně vertikálním pruhem (na wireframu jde o položku „Test execution“).
2. Přepínač, kterým se mění šířka menu (plná velikost nebo zúžená). Web bude rezponzivní, při malé šířce okna se tímto tlačítkem zobrazuje a skrývá celé menu.
3. Hlavička obsahové stránky, na levé straně je název aktuální stránky (v případě definice testu může být stránkou např. přehled testů, detail testu). Na pravé straně se nachází akční tlačítka.
4. Prvek obsahující jméno přihlášeného uživatele, prvek obsahuje rozbalovací menu (nastavení uživatele, odhlášení).
5. Vyjíždějící panel (menu) se seznamem vybraných běhů testů k porovnání. Návrh panelu je na samostatném wireframe 5.8.
6. Akční tlačítka podle obsahu stránky. V případě úzké šířky stránky se tlačítka budou zobrazovat pod sebou a pod názvem stránky.
7. Obsah stránky.

5.7.2 Zadávání dat ve formulářích

Jako vhodná volba se ukázalo zobrazit formuláře pro zadávání dat v modálních oknech, jelikož formuláře pro vytváření a editaci testů, metrik, upozornění nejsou extrémně rozsáhlé. Popis jednotlivých prvků pro wireframe 5.3:

1. Hlavička modální okna s názvem operace, vpravo se nachází křížek pro uzavření okna.
2. Popis prvku formuláře. Všechny popisky jsou zarovnané vpravo.
3. Všechny vstupní prvky formuláře mají nastavenou stejnou šířku.
4. Validační chyby se zobrazují pod příslušným vstupním prvkem.
5. Tlačítka pro odeslání nebo zrušení formuláře. Zarovnání vpravo.

Obrázek 5.2: Návrh rozložení stránky a ovládacích prvků



Obrázek 5.3: Návrh formuláře v modálním okně

5.7.3 Vyhledávání a zobrazení dat

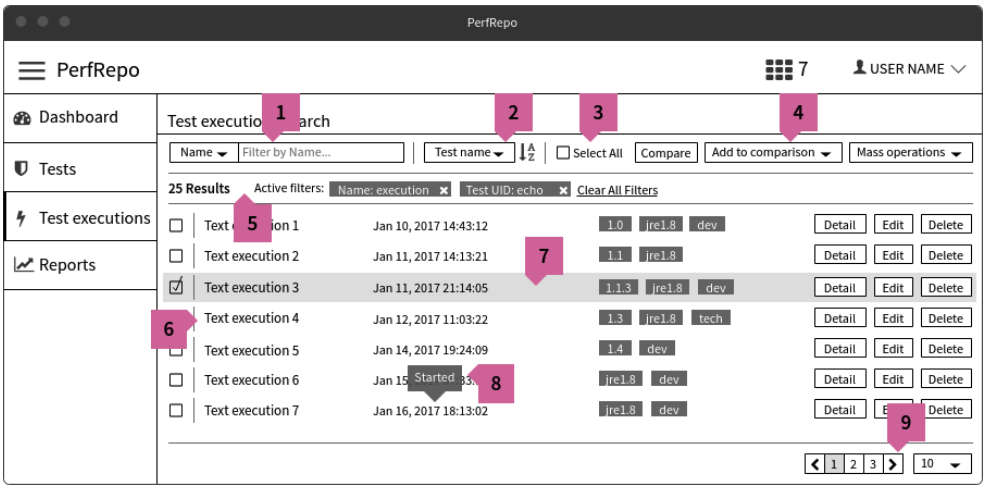
Zobrazení a vyhledávání testů, běhů testů a reportů je provedeno pomocí responzivního seznamu položek. V hlavičce se nachází panel nástrojů pro filtrování, řazení a další operace nad daty. Popis jednotlivých prvků pro wireframe 5.4:

1. Panel pro zadání filtru. Tlačítkem nalevo se z nabídky vybírá typ filtru. Přidání filtru se provede stisknutím klávesy `Enter`.
2. Panel pro nastavení řazení obsahu. Z nabídky se vybírá parametr, podle kterého se má provést řazení. Ikonou vpravo se přepíná pořadí řazení na vzestupné, či sestupné. Ikona se mění podle typu obsahu (text, číslo) a nastaveného pořadí.
3. Zaškrtávací tlačítko pro výběr všech položek seznamu. Při vybrání všech položek mění tlačítko funkci na zrušení výběru.
4. Akční tlačítka pro vybrané položky (porovnání běhů, přidání do seznamu porovnávaných, hromadné operace).
5. Panel s celkovým počtem vyfiltrovaných položek a seznamem aplikovaných filtrů. Filtry lze odstranit jednotlivě nebo najednou.
6. Položka seznamu, vlevo se nachází zaškrtávací tlačítko pro výběr položky. Na pravé straně se nachází akční tlačítka pro aktuální položku.
7. Vizualizace vybrané položky.
8. Položka obsahuje ve sloupcích hodnoty, najetím nad hodnotu se zobrazí popisek s názvem parametru. To je z důvodu, že seznam neobsahuje (oproti tabulce) hlavičku s názvy parametrů.
9. Stránkování seznamu s možností výběru velikosti stránky.

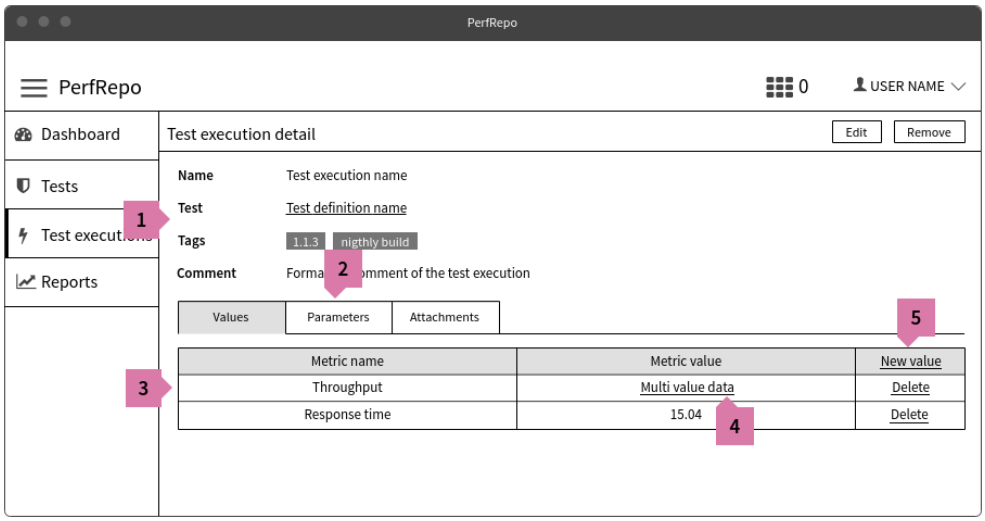
5.7.4 Detail testu a běhu testu

Detail testu a běhu testu je realizovaný stejným způsobem. Popis jednotlivých prvků u běhu testu pro wireframe 5.5:

1. Výpis atributů běhu testu, které lze reprezentovat na jeden řádek.
2. Zobrazení obsahu v panelech (tabs). Běh testu – naměřené hodnoty, parametry spuštění, souborové přílohy. Definice testu – metriky, upozornění (alerts). Prvotní návrh počítal zobrazit všechen obsah pod sebe bez použití panelů. Lepším řešením se ukázalo rozdělit obsah do panelů pro zlepšení přehlednosti.



Obrázek 5.4: Návrh pro vyhledávání a zobrazení běhů testů

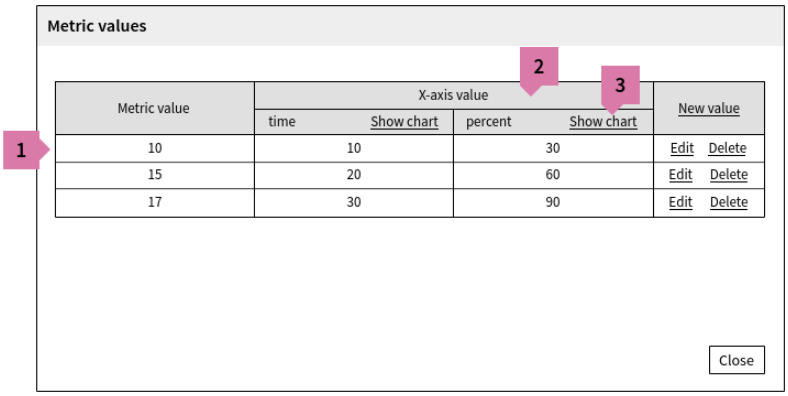


Obrázek 5.5: Návrh detailu běhu testu s naměřenými hodnotami

3. Každý panel obsahuje tabulku s obsahem.

4. Tabulka je přizpůsobena pro zobrazení hodnot metrik s jedním údajem, jelikož naměření sekvence je méně časté. Zobrazení hodnot sekvence je realizováno pomocí odkazu, který vyvolá modální okno, viz 5.6.

5. Sloupec s akcemi, každá tabulka ho obsahuje.



Obrázek 5.6: Návrh zobrazení naměřené sekvence hodnot metriky

5.7.4.1 Zobrazení naměřené sekvence hodnot metriky

Modální okno s naměřenou sekvencí hodnot metriky. Jednotlivé hodnoty je nutné parametrizovat (např. pro účely vykreslení do grafu). Popis jednotlivých prvků pro wireframe 5.6:

1. Položka jedné hodnoty ze sekvence.
2. Sloupce s parametry hodnot (konkrétně na wireframe jsou dva různé parametry), tabulka se dynamicky mění podle počtu parametrů.
3. Sekvenci hodnot lze zobrazit ve spojnicovém grafu (modální okno).

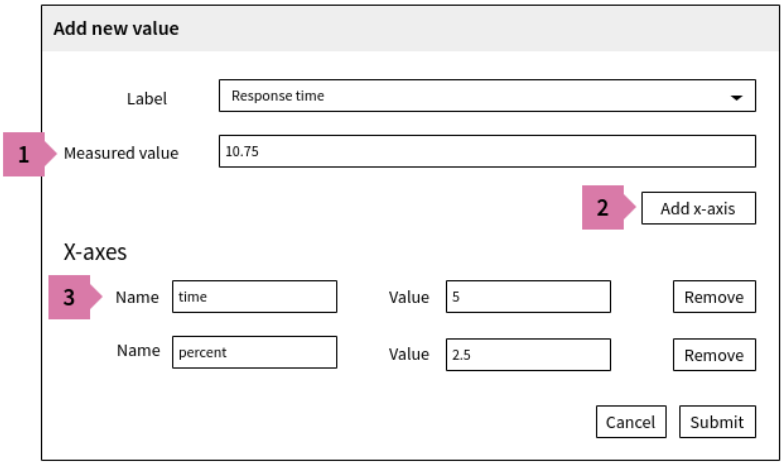
5.7.4.2 Zadání naměřené hodnoty s parametry

Formulář pro zadání hodnoty metriky (včetně parametrů hodnoty). Popis jednotlivých prvků pro wireframe 5.7:

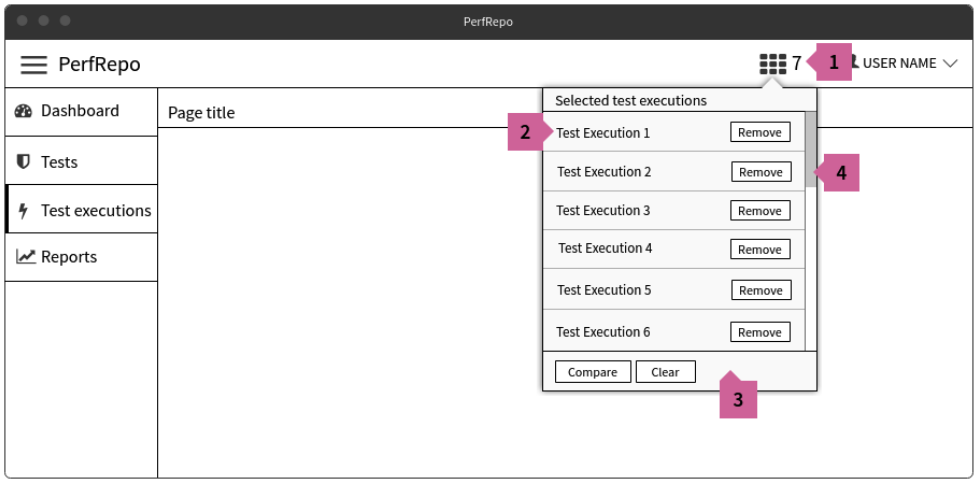
1. Hodnota metriky.
2. Přidání parametru hodnoty.
3. Jednotlivé parametry hodnoty. Tyto parametry mají smysl jen pro naměřené sekvence hodnot.

5.7.5 Menu s výběrem běhů testů k porovnání

Vybrat běh testu pro porovnání lze z vícero míst aplikace (v detailu běhu testu, v seznamů běhů testů nebo v reportu). Vybrané běhy testů jsou dostupné v „popover“ menu, zde je možné vytvořit porovnání (report) nebo odebrat jednotlivé položky. Popis jednotlivých prvků pro wireframe 5.8:

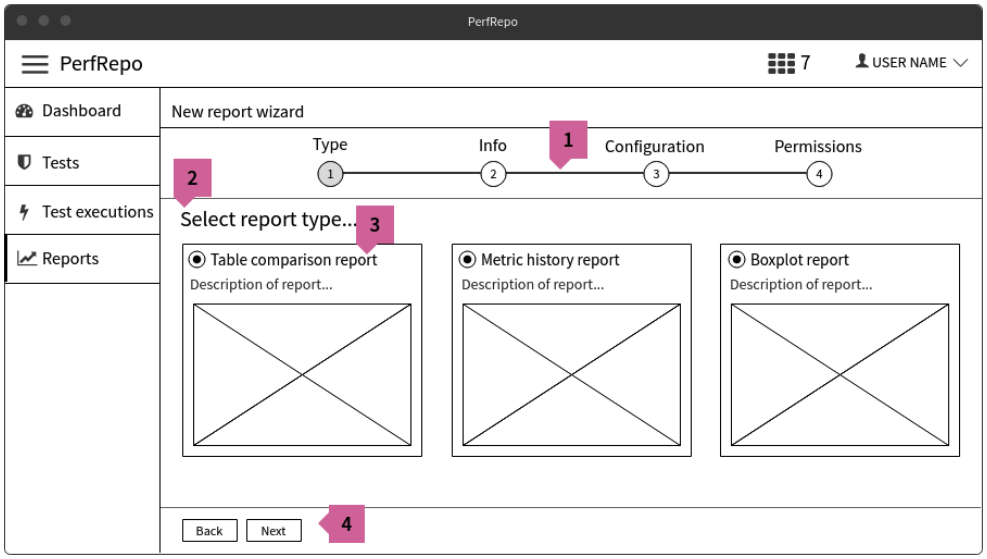


Obrázek 5.7: Návrh zadání naměřené hodnoty s parametry



Obrázek 5.8: Návrh menu s výběrem běhů testů k porovnání

1. Tlačítko pro vyvolání menu, číslice signalizuje počet vybraných položek.
2. Položka seznamu, název běhu testu je odkazem do detailu běhu testu.
3. Patička menu obsahující tlačítka pro smazání všech položek a vytvoření reportu.
4. Posouvací lišta nad obsahem položek.



Obrázek 5.9: Návrh pro výběr typu reportu v průvodci

5.7.6 Průvodce pro vytvoření reportu

Konfigurace reportu je realizována pomocí průvodce se čtyřmi kroky (typ reportu, popis s názvem, konfigurace reportu a nastavení přístupových práv). V hlavičce průvodce se nachází navigace se všemi kroky, aktuální krok je zvýrazněn. Navigace nabízí možnost vrátit se zpět na jakýkoliv krok. Pro přepnutí do dalšího kroku slouží tlačítko v patičce průvodce (přeskakovat kroky bude zakázané kvůli nutné validaci každého kroku).

5.7.6.1 Vybrání typu reportu

Prvním krokem je výběr typu reportu (např. porovnání hodnot, sledování metriky). Popis jednotlivých prvků pro wireframe 5.9:

1. Navigační lišta zobrazuje všechny kroky průvodce se zvýrazněním aktuálního.
2. Popis operace, kterou má uživatel vykonat.
3. Položka typu reportu. Položka obsahuje popis reportu a obrázek typického výstupu.
4. Tlačítka pro přepnutí na další nebo předchozí krok.

5.7.6.2 Konfigurace reportu pro porovnání hodnot

V rozhraní konfigurace reportu pro porovnání hodnot je především nutné vybrat jednotlivé běhy testů k porovnání. Report obsahuje skupiny porovnávání, kde v každé skupině může být několik porovnávacích tabulek. Titulek skupiny a porovnávací tabulky bude dynamicky měněn podle zadaného názvu (ve wireframe „New group“, „New comparison“). Kliknutím na titulek bude moci být obsah minimalizován. Popis jednotlivých prvků pro wireframe 5.10:

1. Skupina porovnání.
2. Konfigurace porovnávací tabulky. Obsah je vlevo odsazen pro přehlednější zobrazení.
3. Výběr běhů testů k porovnání. Změnou selektoru se UI dynamicky mění na potřebná vstupní políčka. UI povolí vybrat pouze jednu položku jako referenční (baseline).
4. Tlačítko pro odstranění skupiny.
5. Tlačítko pro přidání skupiny.

5.7.6.3 Konfigurace reportu pro sledování metrik

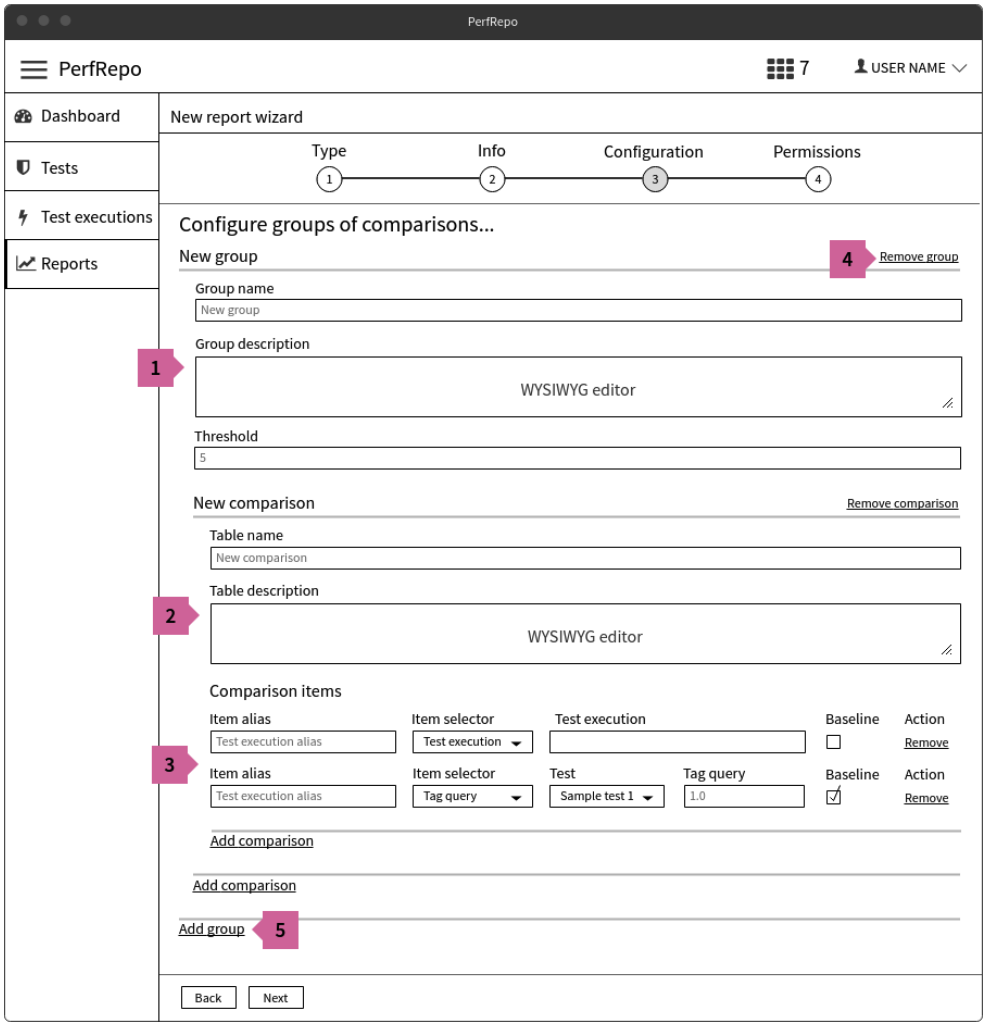
Chování a styl formuláře je stejný jako u předchozího reportu. Popis jednotlivých prvků pro wireframe 5.11:

1. Nastavení názvu a popisu grafu.
2. Výběr jednotlivých sérií v grafu, jedna série představu sledovanou metriku. Filtrem se omezuje výběr běhů testů.
3. Nastavení referenční hodnoty v grafu, výběr konkrétního běhu testu.

5.7.6.4 Konfigurace reportu pro sledování metriky obsahující sekvenci hodnot

Tento typ reportu je určen pro sledování metriky, která obsahuje sekvenci naměřených hodnot. V konfiguraci lze vybrat pouze jednu sledovanou metriku (sérii), výsledek je reprezentován krabicovým grafem. Popis jednotlivých prvků pro wireframe 5.12:

1. Konfigurace krabicového grafu, report může obsahovat více grafů.
2. Nastavení osy grafu. Popis hodnot a řazení (základní hodnota je v obou případech podle času spuštění testu).



Obrázek 5.10: Návrh konfigurace reportu pro porovnání hodnot

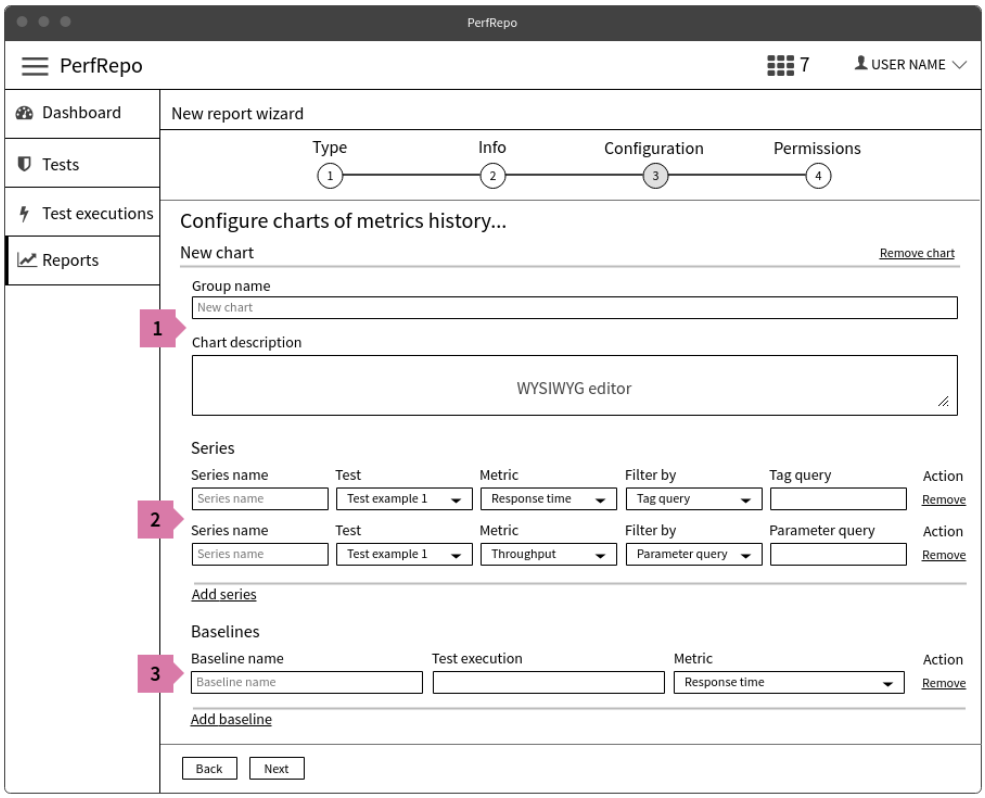
3. Nastavení série (výběr zdrojového testu, metriky a vyfiltrování běhů testů).

5.7.6.5 Nastavení přístupových práv k reportu

Posledním krokem je nastavení přístupových práv k reportu. Popis jednotlivých prvků pro wireframe 5.13:

1. Položka přístupu, vstupní políčka se dynamicky mění podle zadané úrovně oprávnění.

2. Přidání další položky.



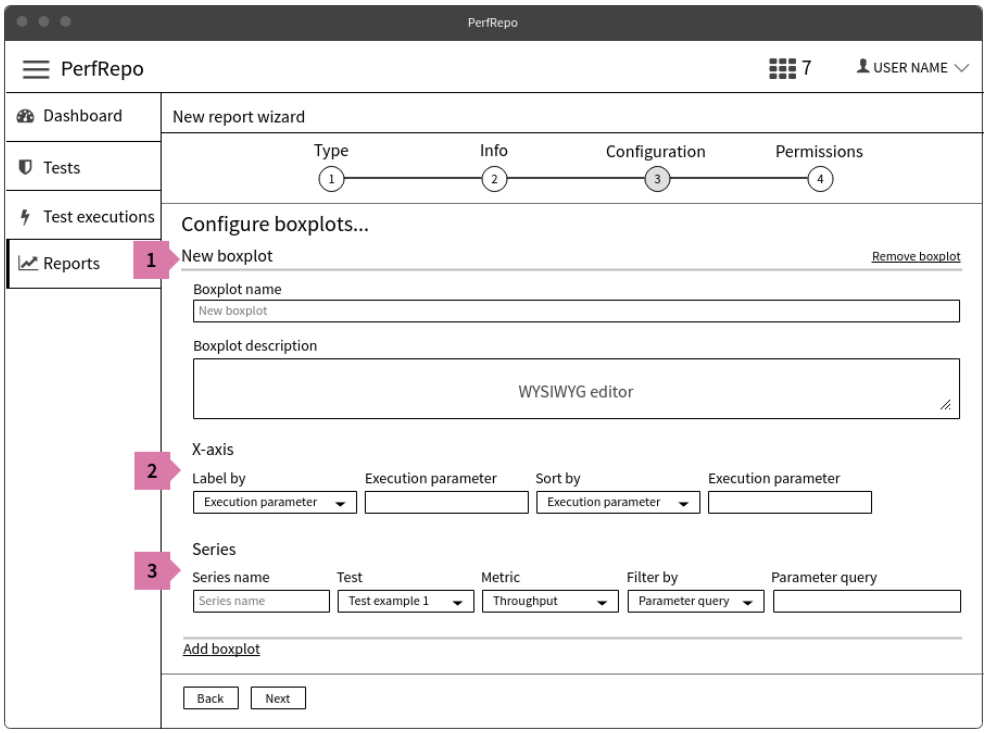
Obrázek 5.11: Návrh konfigurace reportu pro sledování metrik

3. Tlačítko v posledním kroku změní název na „Save“ a slouží pro uložení reportu.

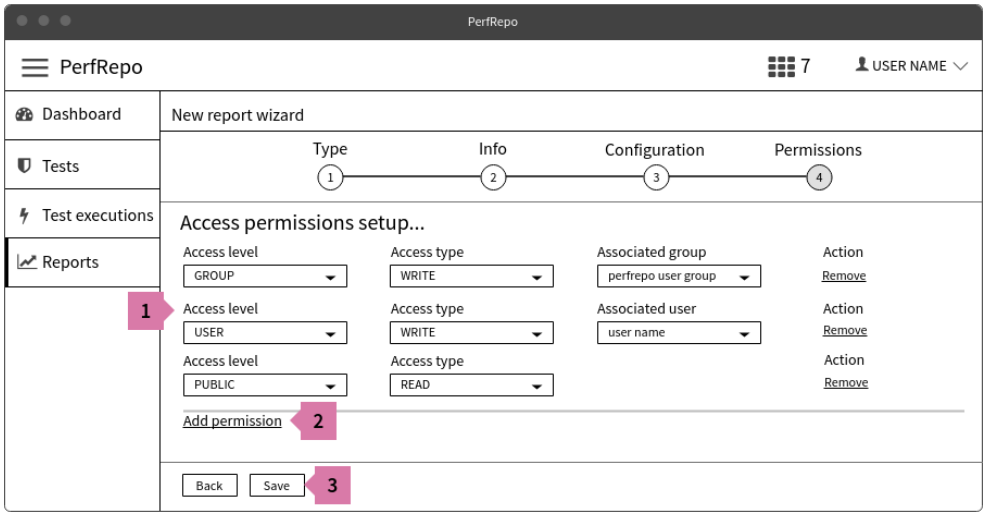
5.7.7 Zobrazení reportů

Report pro porovnání hodnot metrik reprezentuje výsledky v tabulce. V případě, že porovnávaná metrika obsahuje více hodnot, pak je porovnávání reprezentováno v grafu. Popis jednotlivých prvků pro wireframe 5.14:

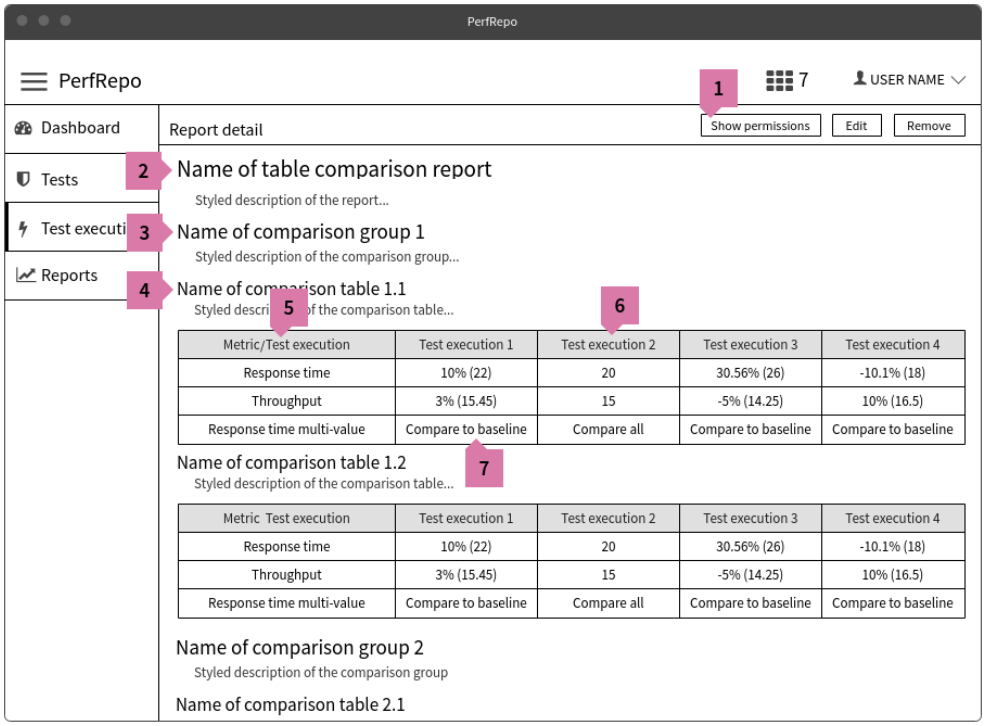
1. Tlačítko pro zobrazení přístupových práv, zobrazení v modálním okně. Původní návrh počítal se zobrazením tabulky s oprávněním přímo v reportu, ale ukázalo se to jako zbytečné.
2. Název reportu s formátovaným popisem.
3. Název skupiny s formátovaným popisem.
4. Název tabulky s formátovaným popisem.



Obrázek 5.12: Návrh konfigurace reportu pro sledování metriky obsahující sekvenci hodnot



Obrázek 5.13: Návrh nastavení oprávnění u reportu



Obrázek 5.14: Návrh zobrazení reportu pro porovnání hodnot

5. Tabulka porovnání. Řádky jsou společné metriky vybraných běhů testů. Sloupce jsou vybrané běhy testů. Název běhu testu je odkazem do detailu běhu testu.

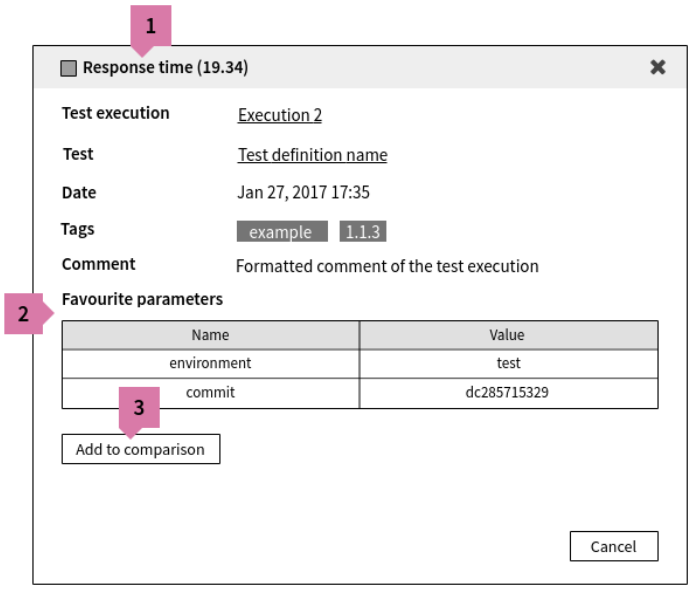
6. Tento běh testu byl zvolen jako referenční, z něho se spočítají procentuální změny naměřených hodnot. Buňky v ostatních sloupcích jsou zabarveny podle rozdílu naměřené hodnoty (zlepšení – zeleně, zhoršení – červeně, změna menší než práh – šedivě).

7. Porovnávaná veličina obsahuje více hodnot, porovnání je vyneseno do grafu (zobrazení v modálním okně).

Jednotlivé body spojnicového grafu a položky krabicového grafu jsou aktivní a po kliknutí se zobrazí modální okno s informacemi o běhu testu, popis jednotlivých prvků pro wireframe 5.15:

1. Název vybrané metriky, v závorce je naměřená hodnota (u krabicového grafu je v závorce medián hodnoty).

2. Parametry běhu testu, které byly označeny jako oblíbené.



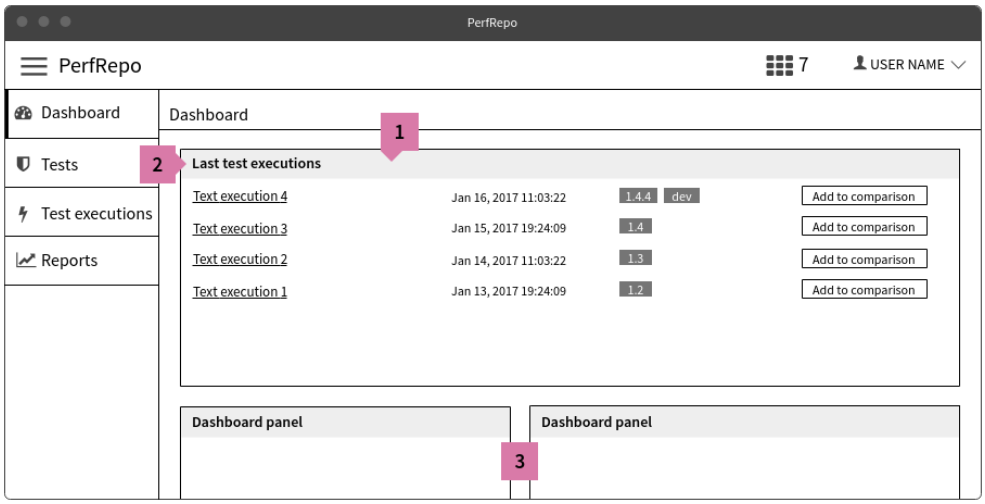
Obrázek 5.15: Návrh zobrazení detailu běhu testu z grafu

3. Tlačítko pro přidání běhu testu k porovnání.

5.7.8 Dashboard stránka

Dashboard stránka je realizována přes jednotlivé panely s informacemi. Popis jednotlivých prvků pro wireframe 5.16:

1. Dashboard panel. Jednotlivé panely se budou skládat pod sebe.
2. Hlavička s názvem panelu. V případě možnosti konfigurace panelu je možné na pravou stranu přidat odkaz na vyvolání nabídky s nastavením.
3. Další panely s menší šířkou.



Obrázek 5.16: Návrh dashboard stránky

Kapitola 6

Implementace

Kapitola se zabývá použitými nástroji a implementací aplikace. Část kapitoly se zabývá popisem struktury a použitými konvencemi.

6.1 Postupy a použité nástroje

Prvním krokem implementace bylo provedení vyčištění projektu od již nepotřebných zdrojových kódů a technologií. Byla smazána celá prezentační vrstva a odstraněny veškeré závislosti a pozůstalosti na technologii JSF.

Po vytvoření základní struktury a rozložení stránky s menu byl vývoj aplikace prováděn po jednotlivých funkčních celcích. Prvním velkým celkem bylo vytváření a správa definic testů a metrik, dalším krokem byly běhy testů, pak následovaly jednotlivé reporty. Pro každý krok byly nejdříve namodelovány DTO objekty (podle funkčních požadavků a potřebám na UI), poté vytvořeno rozhraní servisní vrstvy a webové služby. Až poté probíhala implementace frontend části.

Implementace servisní vrstvy a přepracování datové vrstvy není součástí práce. Tato práce bude prováděna jinými členy komunity. Pro ověření funkčnosti uživatelského rozhraní musela být během vývoje vytvořena provizorní implementace, která slouží pouze pro testovací účely. Není řešeno perzistentní ukládání dat (vytvořená data jsou při běhu aplikace uchovávána v kolekcích), mnohé metody jsou implementovány jen částečně a vrací umělá data (metoda mockování).

6.1.1 Verzování a správa projektu

Projekt je verzovaný nástrojem Git a hostovaný na serveru GitHub. Adresa repozitáře je `https://github.com/PerfCake/PerfRepo`. Pro vývoj nové verze byl vytvořený tzv. „fork“ projektu. V hlavním repozitáři byla vytvořena nová

větev pro novou verzi aplikace, do které byly po dokončených celcích posílány změny ke schválení a zaintegrování (tzv. „pull request“). Název vývojové větve je `2.0_devel_frontend`.

6.1.2 Vývojové prostředí

Pro vývoj bylo použito vývojové prostředí IntelliJ IDEA. Z vyzkoušených vývojových prostředí (Netbeans IDE, Eclipse) má IntelliJ IDEA nejlepší podporu Javasciptu a AngularJS.

6.1.3 Použité nástroje pro sestavení aplikace

Pro sestavení frontend části aplikace byl použit nástroj Gulp. Vytvořený build se stará o stažení závislostí (knihovny), překlad kaskádových stylů ze SASS do CSS, minifikaci kódu, spojení několika souborů zdrojového kódu do jednoho souboru. Důležitým úkolem je automatické linkování skriptů a stylů do `index.html` souboru a to včetně vyřešení pořadí linkování jednotlivých skriptů. Automatické linkování velmi usnadňuje vývoj.

Konfigurační Gulp soubor pro sestavení aplikace obsahuje jednotlivé úlohy (např. zpracování skriptů knihoven, zpracování skriptů aplikace, překlad stylů ze SAAS do CSS). Spojením těchto úloh vznikla hlavní úloha, která slouží pro sestavení aplikace. Užitečnou úlohou je úloha s názvem `watcher`, která se používá při vývoji aplikace. Během spuštění této úlohy jsou sledovány změny ve zdrojovém kódu a spouštěny potřebné úlohy (např. překlad stylů, nalinkování nového souboru). Úloha se spouští příkazem `gulp watcher`.

Na serverové části je použit nástroj Maven[17]. Ten při sestavení aplikace spouští i samotný build pro frontend část[18] Výsledkem je webový archiv WAR.

6.2 Struktura projektu a použité konvence

Následující kapitola popisuje strukturu projektu a jeho rozdělení do modulů. V další části jsou pak popsané přejaté konvence pro psaní AngularJS aplikací.

Na serverové části aplikace je nasazený nástroj, který kontroluje styl kódu napsaného v jazyce Java. Ten je velmi striktní, upozorňuje na potenciální chyby, ale i na špatné formátování zdrojového kódu.

[17] Maven – systém pro správu a sestavování aplikací pod platformou Java

[18] Pro sestavení frontend části je zapotřebí mít dostupný balíčkový systém NPM[31] a framework Node.js[32]. Není potřeba nic řešit, o vše se postará nakonfigurovaný Maven build.

6.2.1 Struktura a členění projektu

Aplikace PerfRepo je rozdělena do modulů `model`, `client` a `web`. Každý tento modul je samostatným Maven artefaktem.

V modulu `model` se nachází umělý model aplikace, tzv. DTO[19] objekty. Lze to pojmenovat i jako veřejný model aplikace, webové služby a servisní vrstva pracuje s těmito strukturami.

Modul `client` je Java klientem pro volání webových služeb aplikace PerfRepo. Umožňuje snadno vytvořit dotaz a odeslat HTTP požadavek. Postará se o serializaci objektů na JSON a o následnou deserializaci. Dále zpracovává HTTP stavové statusy, které v případě chyby interpretuje na výjimky (Java exception). Tento modul má závislost na modul `model`. Implementace tohoto modulu není součástí práce.

Modul `web` obsahuje webovou aplikaci a má závislost na modulu s modelem. Souborová struktura je následující (zjednodušeno):

```
src ..........................................................
  main ..... zdrojové kódy serverové části aplikace
    antlr3 ..... definice gramatik[20]
    java ..... zdrojové kódy v jazyce Java
    resources ..... soubory s konfiguračními a lokalizačními daty
    webapp ..... soubory webové aplikace
      WEB-INF ..... konfigurační soubory
  test ..... testy serverové části aplikace
target ..... vygenerované výstupní soubory, obsahuje i WAR soubor
webapp_content ..... zdrojové kódy AngularJS aplikace
bower.json ..... závislosti webové aplikace
gulpfile.js ..... Gulp konfigurace pro build aplikace
package.json ..... závislosti pro build aplikace
```

6.2.2 Přejaté konvence pro vývoj aplikací v AngularJS

Konvence pro psaní aplikací v AngularJS byly převzaty od již zmiňovaného John Papa. Konvence popisují strukturu aplikace, vhodné pojmenování souborů, tvorbu modulů, formátování zdrojového kódu.

Každý soubor zdrojového kódu kromě kaskádových stylů je vhodně pojmenován podle vzoru `<nazev_souboru>.<typ>.<format>`. Formátem je klasická přípona souboru (js, html). Typ je logická část AngularJS aplikace,

[19] DTO (Data transfer object) – objekt určený pro přenos dat, objekt neobsahuje žádnou business logiku

typy vyskytující se v aplikaci jsou: `controller`, `module`, `service`, `config`, `interceptor`, `directive`, `filter`, `factory` a `view` (html šablona).

Další konvencí je mít pro každou direktivu, komponentu, filtr a další typy samostatný soubor. Nový modul vždy definovat v novém souboru mimo direktivu nebo komponentu.

Velkým neduhem u aplikací v AngularJS je manipulace s DOMem stránky z řídící vrstvy (`controller`). To ani není úplně konvence, ale úplně špatné použití AngularJS, aplikace se stane nepřehlednou, špatně testovatelnou a prezentační vrstva se stane silně závislou na řídící vrstvě. Velmi častá chyba vývojářů začínajících v AngularJS. Vhodným řešením je napsaní direktivy, která rozšíří možnosti elementu.

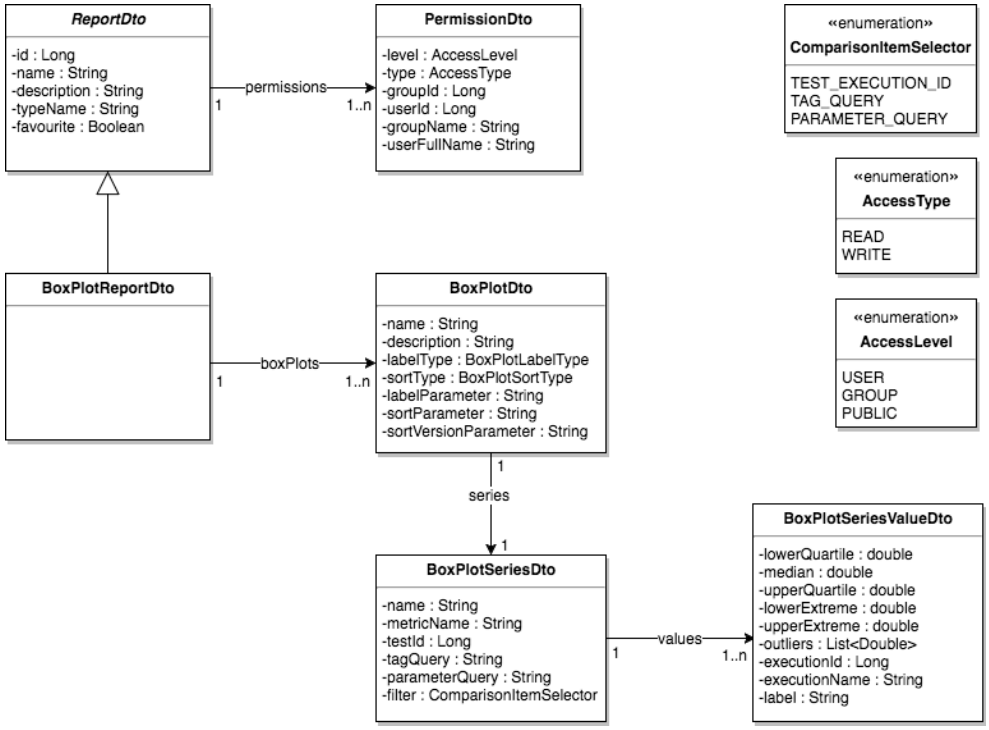
6.3 Datový model

Jedním z požadavků bylo zapouzdřit datový model (entitní třídy) aplikace. Od servisní vrstvy se pracuje s DTO objekty. Servisní vrstva provádí konverzi mezi těmito modely. Důvodem pro zavedení tohoto umělého modelu bylo:

- Přizpůsobit strukturu dat pro potřeby webové aplikace.
- Databázový model aplikace je velmi abstraktní. Například pro všechny typy reportů je jedna stejná datová struktura, která je řešena pomocí skupin dvojic (klíč, hodnota).
- Manipulace s databází probíhá přes relační mapování (ORM) a odstíněním se vyvarujeme mnohým chyb (hlavně vznik výjimky `LazyInitializationException`).
- Zajistí se snadnější přechod na novou verzi aplikace. Existující databáze bude kompatibilní s novou verzí (pouze rozšíření modelu a případné menší změny).
- DTO objekty obsahují pouze datové atributy bez business logiky. To je výhodné při oddělení těchto struktur do samostatného modulu.

6.3.1 Datová struktura reportů

Každý report má navrženou speciální datovou strukturu pro snadné vytvoření a zobrazení na webu. Vypočítání hodnot do reportu a sestavení dat pro grafy provádí servisní vrstva na serverové části. Všechny typy reportů dědí od třídy `ReportDto`.



Obrázek 6.1: Třídní model „BoxPlot“ reportu

Na diagramu 6.1 jedna z mála částí datového modelu, zde je konkrétně třídní model reportu pro sledování metriky obsahující sekvenci hodnot. V diagramu kvůli šetření místa chybí metody pro získání a nastavení atributů třídy (tzv. „getter“ a „setter“ metody).

6.4 Rozhraní pro servisní vrstvu

Rozhraní servisní vrstvy je reprezentováno klasickými Java rozhraními. Kontraktem rozhraní jsou DTO objekty. Entitní třídy (model aplikace) jsou z pohledu servisní vrstvy zapouzdřené a servisní vrstva provádí mezi těmito typy konverzi. Názvy rozhraní jsou `AlertAdapter`, `AuthenticationAdapter`, `ComparisonSessionAdapter`, `GroupAdapter`, `MetricAdapter`, `ReportAdapter`, `TestAdapter`, `TestExecutionAdapter` a `UserAdapter`.

Jak již bylo řečeno, implementace servisní vrstvy není součástí práce, už jen z toho důvodu bylo nutné vytvořit vypovídající dokumentaci pro rozhraní servisní vrstvy, popsat struktury a formát objektů se statusy validace. Dokumentace je tvořena pomocí nástroje Javadoc.

6.4.1 Propagace chyb ze servisní vrstvy

Při volání služeb může na servisní vrstvě docházet k chybám. Může se jednat pouze o chybu nevalidního dotazu nebo také chyby v programu. Tyto chyby by měla implementace servisní vrstva propagovat přes skupinu vyspecifikovaných výjimek. V následujících podkapitolách jsou popsány jednotlivé výjimky.

6.4.1.1 Chyba validace

Chyba validace je reprezentovaná třídou `ValidationException`. Tato výjimka je vyvolána, pokud dotaz obsahuje nevalidní data. K této chybě typicky dojde při vytváření nebo úpravě dat v aplikaci. Třída výjimky obsahuje seznam vzniklých chyb. Chyba je reprezentována názvem a zprávou chyby.

6.4.1.2 Chyba oprávnění

Chyba je vyvolána, pokud uživatel nemá dostatečné oprávnění k provedené požadovaného volání. Chyba je reprezentovaná třídou `UnauthorizedException`.

6.4.1.3 Chyba nenalezení entity

Při dotazech na entity, jejíž instance v systému neexistují, se vyhodí výjimka o nenalezení požadovaného zdroje. Chyba je reprezentována třídou `NotFoundException`.

6.4.1.4 Univerzální chyba špatného dotazu

Tato chyba slouží k určení špatného volání. Dotaz je nevalidní a nemohl být proveden. Chyba je reprezentovaná třídou `BadRequestException`.

6.4.1.5 Chyba servisní vrstvy

Nejzávažnější chyba, slouží pro indikaci chyby na servisní vrstvě. Chybou může být například NPE[21], ztráta připojení k databázi. Chyba je reprezentovaná třídou `AdapterException`.

6.5 Webové REST služby

Pro novou verzi aplikace byly kompletně přepracovány REST webové služby. Preferovaným datovým formátem je JSON, ale malými změnami v konfiguraci lze použít i XML. Služby pro účely webové aplikace a pro API aplikace jsou společné. Seznam s popisem služeb je dostupný z `https://github.com/grunwjir/PerfRepo.v2/wiki/REST-API`.

[21] NPE – Null Pointer Exeception

Tabulka 6.1: Použité HTTP statusy ve webových službách aplikace

HTTP status	**Použití v aplikaci**
200 OK	požadavek korektně zpracován
201 Created	vytvoření nového zdroje, služba vrací hlavičku `Location`
204 No Content	požadavek zpracován, nevrací se žádná data
400 Bad Request	špatný dotaz, nelze zpracovat
401 Unauthorized	chyba nepřihlášení uživatele
403 Forbidden	nepovolená akce pro přihlášeného uživatele
404 Not Found	zdroj nenalezen
422 Unprocessable Entity	vytvářená nebo upravovaná entita obsahuje chyby validace
500 Server Error	chyba serveru

Služby jsou navrženy podle konceptu RESTful, využívají vhodné HTTP metody. Služby vracejí následující stavové kódy:

Webové služby používají typické HTTP statusy, viz tabulka 6.1. Za úvahu stálo, jaký použit status pro požadavek, který obsahuje nevalidní data, ale není chybný. Jako vhodný se jeví status kód 422 (Unprocessable Entity).

6.5.1 Serializace a deserializace objektů

Objekty jsou serializovány do formátu JSON pomocí knihovny Jackson. Serializace a deserializace funguje prakticky automaticky bez většího úsilí[22], jen byla provedena změna serializace pro datum s časem do normy ISO 8601 a musela být vyřešena deserializace polymorfních typů.

Knihovna Jackson serializuje datum s časem jako počet milisekund od roku 1970 (tzv. Unixová epocha). Pro lepší čitelnost byla serializace změněna na formát ISO 8601 a čas je vyjádřen ve světovém čase (UTC). Změna je provedena pomocí jednoduché změny mapování, viz ukázka zdrojového kódu 6.1. Toto nastavení je aplikováno na všechny poskytované webové služby.

Zdrojový kód 6.1: Změna serializace datumu s časem do normy ISO 8601

```
@Provider
@Produces(MediaType.APPLICATION_JSON)
public class JacksonConfig implements ContextResolver<ObjectMapper> {
    private final ObjectMapper objectMapper;

    public JacksonConfig() throws Exception {
```

[22] DTO objekty se většinou mapují jedna ku jedné a není potřeba další konfigurace

```
        objectMapper = new ObjectMapper();
        objectMapper.configure(
         SerializationFeature.WRITE_DATES_AS_TIMESTAMPS,
            false);
    }

    @Override
    public ObjectMapper getContext(Class<?> arg) {
        return objectMapper;
    }
}
```

Úryvek zdrojového kódu 6.2 ukazuje nastavení serializace a deserializace polymorfního typu. Jednotlivé typy reportů dědí od abstraktní třídy `ReportDto`. Aby při deserializaci mohlo být rozhodnuto, o jaký typ reportu se jedná, musí se při serializaci tato informace přidat. To je řešené pomocí anotací a do serializovaného objektu se přidává atribut `type` s hodnotou `TABLE_COMPARISON`, `METRIC_HISTORY` nebo `BOX_PLOT` podle typu reportu.

Zdrojový kód 6.2: Nastavení abstraktního typu pro serializaci a deserializaci objektu

```
@JsonTypeInfo (
        use = JsonTypeInfo.Id.NAME,
        include = JsonTypeInfo.As.PROPERTY,
        property = "type")
@JsonSubTypes({
        @JsonSubTypes.Type(value = TableComparisonReportDto.class,
            name = "TABLE_COMPARISON"),
        @JsonSubTypes.Type(value = MetricHistoryReportDto.class, name
            = "METRIC_HISTORY"),
        @JsonSubTypes.Type(value = BoxPlotReportDto.class, name =
            "BOX_PLOT")})
public abstract class ReportDto {
    private Long id;
    private String name;
    private String description;
    private String typeName;
    private List<PermissionDto> permissions;

   // ... getter & setter methods
}
```

6.5.2 Zpracování výjimek ze servisní vrstvy

Dohodnuté výjimky ze servisní vrstvy jsou tzv. nekontrolované (unchecked) a jsou automaticky odchyceny a zpracovány. Každá druh výjimky je namapo-

vaný na obslužný kód. Ten určuje strukturu a stavový kód odpovědi HTTP požadavku. Ukázka takového zpracování je v úryvku zdrojového kódu 6.3, ve kterém je serializována výjimka při nenalezení požadovaného zdroje.

Zdrojový kód 6.3: Ukázka serializace výjimky

```
@Provider
public class NotFoundExceptionMapper implements
    ExceptionMapper<NotFoundException> {
    @Override
    public Response toResponse(NotFoundException exception) {
        Map<String, Object> message = new HashMap<>();
        message.put("message", exception.getMessage());
        message.put("source", "NOT_FOUND EXCEPTION");
        return Response
         .status(Response.Status.NOT_FOUND)
         .entity(message)
         .build();
    }
}
```

6.5.3 Zabezpečení webových služeb

Pro úspěšné zavolání webových služeb je nutné, aby se volající autentizoval. To je zde zajištěno pomocí přístupového `Bearer` tokenu. V každém volání služby, která je zabezpečená, je nutné do HTTP hlavičky `Authorization` přiložit tento přístupový token. Před hodnotu tokenu je nutné přiložit typ autorizace, příkladem použití je např. `Authorization: Bearer BxCdEf123456`.

Volající získá přístupový token po úspěšném zavolání služby pro autentizaci. V aplikaci je získání tokenu možné z URL adresy `/api/json/authentication`. Detaily o způsobu generování tokenů a jejich expiraci je na implementaci servisní vrstvy a diplomová práce se jím nezabývá. V budoucnu se nabízí možnost rozšíření o externího autentizačního poskytovatele (např. přes OpenID nebo Google účet).

6.6 Prezentační část aplikace

Prezentační částí je myšlena AngularJS aplikace, to ve skutečnosti představuje aplikaci běžící na straně klienta, která má svůj model, servisní, řídící a prezentační vrstvu. V kapitole jsou popsány použité knihovny a úryvky zdrojového kódu.

6.6.1 Použité AngularJS komponenty třetích stran

Většina základních komponent pro vytvoření uživatelského rozhraní je již obsažena ve frameworcích UI Bootstrap a PatternFly. Pro některé specifické potřeby ale musely být použity komponenty a funkcionality třetích stran.

Výčet použitých knihoven pro rozšíření funkcionality a nabídky komponent frameworku AngularJS:

UI Bootstrap[23]
: Komponenty frameworku Bootstrap (modální okno, stránkování, panel s taby, tooltip).

AngularJS PatternFly[24]
: Komponenty frameworku PatternFly, část komponent používá komponenty z UI Bootstrap (menu, wizard, filtrování a zobrazení dat, notifikacey).

AngularUI Router[25]
: Komponenta pro routování stránek, základ pro SPA[26].

ngProgress[27]
: Komponenta sloužící pro zobrazení informace, že dochází k načítaní obsahu. V horní části stránky se zobrazuje úzký pruh se zvyšující se šířkou až k úplné šířce stránky.

Angular-trix[28]
: WYSIWYG editor. Použití pro zadávání všech popisů, které mají být formátovatelné.

ngTagsInput[29]
: Komponenta pro zadávání štítků. Do jednoho vstupního textového pole se zadávají jednotlivé štítky, které jsou automaticky oddělené na samostatné tokeny. Použití pro zadávání štítků u běhů testů.

Multi-level accordion[30]
: Komponenta pro vytvoření panelu s rozsáhlým obsahem, který lze vertikálně minimalizovat. Použití ve wizardu reportu.

[23] Dostupné z `https://github.com/angular-ui/bootstrap`
[24] Dostupné z `https://github.com/patternfly/angular-patternfly`
[25] Dostupné z `https://github.com/angular-ui/ui-router`
[26] Single Page Application
[27] Dostupné z `https://github.com/VictorBjelkholm/ngProgress`
[28] Dostupné z `https://github.com/sachinchoolur/angular-trix`
[29] Dostupné z `https://github.com/mbenford/ngTagsInput`
[30] Dostupné z `https://github.com/LukaszWatroba/v-accordion`

UI Select[31]
: Komponenta pro pokročilé výběrové rozbalovací menu, které nabízí filtrování nabízených položek. Nabízené položky je možné dynamicky načítat ze serverové části podle zadaného filtru. Použití pro výběr metrik nebo například pro výběr definice testu ve wizardu vytvářeného reportu.

Angular-nvD3[32]
: Komponenta pro vykreslování grafů. Výhodou této knihovny je podpora krabicového grafu (boxplot).

6.6.2 Volání webových služeb

Volání webových služeb je prováděno prostřednictvím AngularJS servisních objektů. Metody zaregistrovaného servisního objektu lze snadno použít v ostatních komponentách aplikace. Stačí přidat název zaregistrovaného servisního objektu do parametrů funkce (to funguje za pomoci techniky Dependency injection).

Na ukázce kódu 6.4 je část servisního objektu, který slouží pro volání webových služeb týkající se definice testu. Ukázka obsahuje pouze jednu metodu pro vyhledávání definic testů, z příkladu je vidět zpracování HTTP hlaviček odpovědi.

Zdrojový kód 6.4: Ukázka servisního objektu pro volání webových služeb

```
(function() {
    'use strict';

    angular
        .module('org.perfrepo.test')
        .service('testService', TestService);

    function TestService($http, $resource, API_TEST_URL) {
        ...
        function search(searchParams){
            return $http.post(API_TEST_URL + '/search',
                searchParams).then(function(response) {
                return {
                    data : response.data,
                    totalCount : parseInt(
                        response.headers('X-Pagination-Total-Count')),
                    pageCount : parseInt(
                        response.headers('X-Pagination-Page-Count')),
                    currentPage : parseInt(
                        response.headers('X-Pagination-Current-Page'))
```

[31] Dostupné z `https://github.com/angular-ui/ui-select`
[32] Dostupné z `https://github.com/krispo/angular-nvd3`

```
                };
            });
        }
        ...
    }
})();
```

U každého autorizovaného volání webové služby je nutné do HTTP hlavičky přidat přístupový `Bearer` token. To v aplikaci řeší tzv. `interceptor`, který se vykoná při každém volání a přidá požadovaný token. Použití je na ukázce kódu 6.5. Stejný princip je použit i pro HTTP odpovědi, kde se kontroluje status kód (např. pro status kód 401 dojde k přesměrování na stránku s přihlášením, pro status kódu 400 se zobrazí notifikace s chybovou zprávou). Přístupový token je uložen v lokálním úložišti webového prohlížeče (`Local Storage`).

Zdrojový kód 6.5: Ukázka přidání přístupového tokenu do hlavičky HTTP požadavku

```
(function() {
    'use strict';

    angular
        .module('org.perfrepo.authentication')
        .config(config);

    function config($httpProvider) {
        $httpProvider.interceptors.push(tokenInterceptor);
    }

    function tokenInterceptor($injector) {
        return {
            request: function(config) {
                var authenticationService =
                    $injector.get('authenticationService');
                if (authenticationService.isAuthenticated()) {
                    config.headers = config.headers || {};
                    config.headers['Authorization'] = "Bearer " +
                        authenticationService.getToken();
                }

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

6.6.3 Nalinkování kaskádových stylů a skriptů do aplikace

Soubory se skripty a kaskádovými styly jsou do webové stránky `index.html` linkovány automaticky pomocí sestavovacího nástroje Gulp. Místo pro vložení určují HTML komentáře se speciální syntaxí. Použití je v ukázce 6.6.

Zdrojový kód 6.6: Způsob nalinkování stylů a skriptů do aplikace

```
<!DOCTYPE html>
<html ng-app="org.perfrepo">
<head>
    <meta charset="utf-8"/>
    <title ng-bind="title">PerfRepo</title>
    <link href="/images/favicon.ico" rel="icon" type="image/x-icon" />
    <!-- inject:css -->

    <!-- endinject -->
</head>
<body>
    <div class="pr-notification-list">
        <pf-notification-list></pf-notification-list>
    </div>
    <ui-view></ui-view>
    <!-- inject:js -->

    <!-- endinject -->
</body>
</html>
```

Kapitola 7

Demonstrace nové verze

Poslední kapitola se zabývá demonstrací schopností nové verze aplikace a ukázkou uživatelského rozhraní. Kapitola se také zabývá nahráváním výsledků testování z nástroje PerfCake do aplikace PerfRepo.

Postup pro instalaci aplikace (nastavení aplikačního serveru, kompilace a nasazení) je na adrese projektu `https://github.com/PerfCake/PerfRepo`. Již zkompilované zdrojové kódy jsou k dispozici na přiloženém CD médiu.

7.1 Nahrání dat z nástroje PerfCake

Naměřené výsledky z nástroje PerfCake se nejsnadněji nahrají do PerfRepo pomocí PerfRepo destination pluginu[33]. Tento plugin má několik nedostatků, a proto je aktuálně preferovaným způsobem nahrávání zpracovaných výsledků do PerfRepo pomocí jeho REST API. Nedostatkem je například nemožnost reportovat více než jednu metriku testu. Vývoj nové verze pluginu je zadán jako bakalářská práce a měl by rozšířit možnosti a zpříjemnit propojení obou nástrojů. Nový plugin by měl umožnit nahrávat výsledky bez předem vytvořené definice testu v aplikaci PerfRepo (plugin vše vytvoří voláním API aplikace).

Změnou webových služeb a datové struktury u nové verze PerfRepo se stal tento plugin nefunkční, pro demonstraci použití byl tento plugin upraven. Navíc byla přidána podpora pro nahrání dat více metrik. Plugin je dostupný z repozitáře `https://github.com/grunwjir/Plugins` (větev `perfrepo2`).

7.1.1 Instalace pluginu do PerfCake

Instalace pluginu do nástroje PerfCake je snadná, stažený plugin se přeloží pomocí příkazu `mvn install`. Pak stačí překopírovat vytvořený JAR soubor

[33] PerfRepo destination plugin je dostupný z `https://github.com/PerfCake/Plugins`

do adresáře `$PERFCAKE_HOME/lib/plugins` a obsah složky `lib` (plugin používá knihovnu Jackson) do adresáře `$PERFCAKE_HOME/lib/plugins/ext`. Nyní je možné pro odesílání výsledků použít destinaci s názvem `PerfRepoDestination`.

7.1.2 Spuštění ukázkového testu

Ukázkový testovací scénář 7.1 měří čas odezvy získání odpovědi ze serveru přes protokol HTTP. Plugin vytvoří v aplikaci PerfRepo nový běh testu a nahraje naměřený čas odezvy serveru (průměrnou, minimální a maximální hodnotu). Test se spustí příkazem `./perfcake.sh -s http_scenario -Dserver.host=google.cz`, kde `http_scenario` je název testovacího scénáře a parametr `-Dserver.host` určuje adresu serveru k testování.

7.2 Ukázka výsledného uživatelského rozhraní

Podkapitola obsahuje ukázku nového uživatelského rozhraní prostřednictvím snímků obrazovky. Ukázána je pouze část aplikace s hlavními prvky, více snímků obrazovky si lze prohlédnout na přiloženém CD médiu[34]. Nutno podotknout, že statické snímky nemůžou plně vyjádřit možnosti a celkové chování aplikace, k tomu je nutné aplikaci vyzkoušet.

7.2.1 Detail definice testu

Detail definice testu je zobrazen na snímku 7.1. Zobrazená definice testu obsahuje tři metriky, nadefinovaná upozornění jsou skrytá ve druhém panelu. Při vyvolání editace definice testu nebo akce pro přidání nové metriky se zobrazuje modální okno. Kliknutím na tlačítko „Show test executions“ dojde k přesměrování na stránku s vyhledáváním běhů testů (přednastaví se filtr pro aktuální definici testu).

7.2.1.1 Vytvoření podmínky pro metriku

Na snímku 7.2 je formulář pro vytvoření podmínky pro metriky. Při nesplnění podmínky je uživatel na tuto událost upozorněn. Syntaxe pro zápis podmínky je na adrese `https://github.com/PerfCake/PerfRepo/wiki/Alerting`. V aplikaci je na to odkázáno.

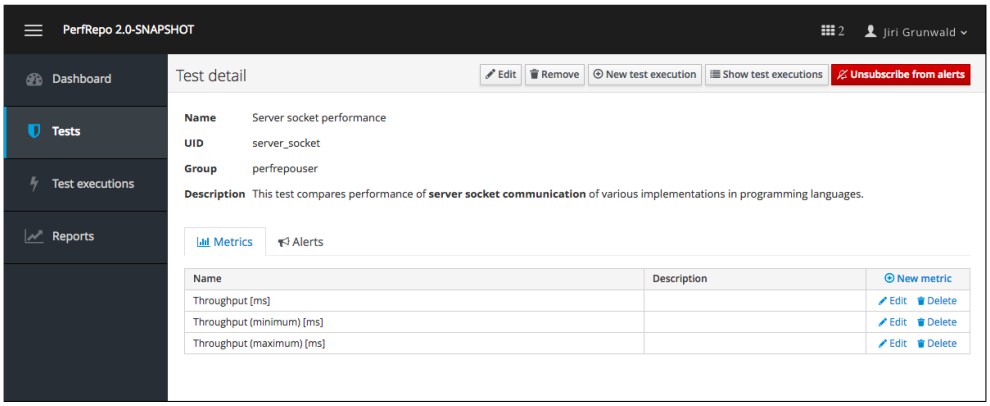
7.2.2 Detail běhu testu

Na snímku 7.3 je zobrazena stránka s detailem běhu testu. Běh testu obsahuje hodnoty tří metrik. Pokud metrika obsahuje více naměřených hodnot,

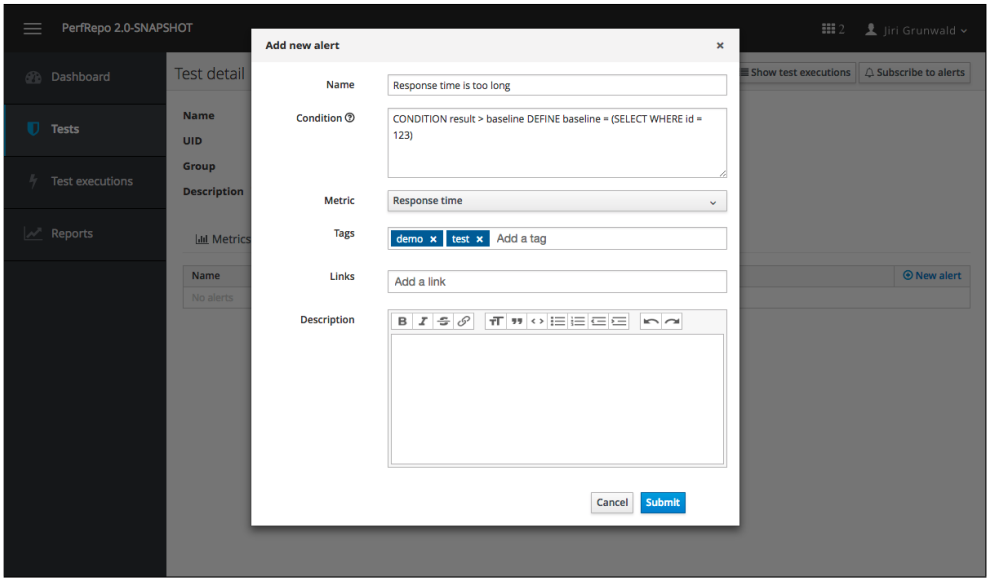
[34] Snímky obrazovky je možné stáhnout i z adresy `https://drive.google.com/drive/folders/0B1OjHNgsiHD7UXhiRUpIaXBjNkE`

Zdrojový kód 7.1: Konfigurace testovacího scénáře s odesíláním výsledků do aplikace PerfRepo

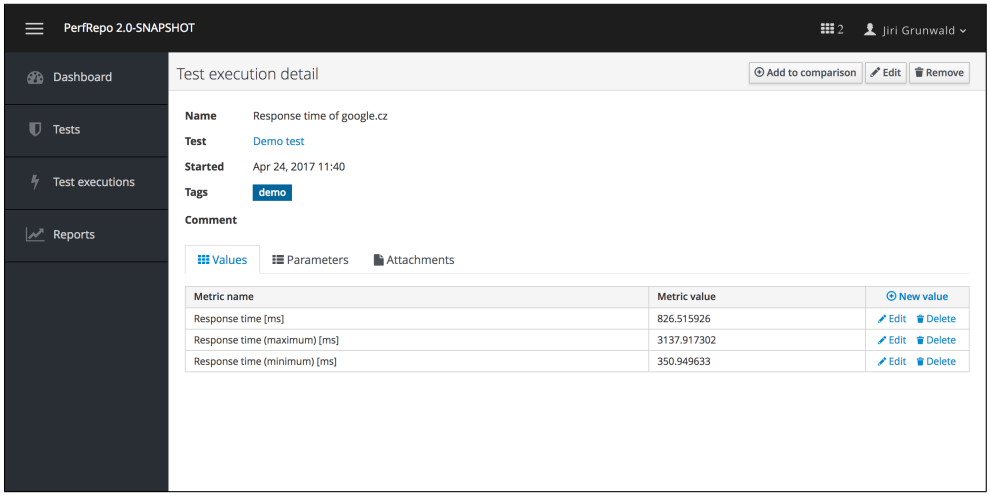
```
<?xml version="1.0" encoding="utf-8"?>
<scenario xmlns="urn:perfcake:scenario:7.0">
   <run type="${perfcake.run.type:time}"
       value="${perfcake.run.duration:5000}"/>
   <generator class="DefaultMessageGenerator"
       threads="${perfcake.thread.count:100}"/>
   <sender class="HttpSender">
      <target>http://${server.host}</target>
      <property name="method" value="GET"/>
   </sender>
   <reporting>
      <reporter class="ResponseTimeStatsReporter">
         <destination class="ConsoleDestination">
            <period type="time" value="1000"/>
         </destination>
         <destination class="PerfRepoDestination">
            <period type="time" value="${perfcake.run.duration:5000}"/>
            <property name="metrics" value="Response time
                [ms];Response time (minimum) [ms];Response time
                (maximum) [ms]"/>
            <property name="resultNames"
                value="Average;Minimum;Maximum"/>
            <property name="repositoryUrl"
                value="http://localhost:8080"/>
            <property name="username" value="grunwjir"/>
            <property name="password" value="123456"/>
            <property name="testUID" value="demo_test"/>
            <property name="tags" value="demo"/>
            <property name="parameters" value="source=local;
               threads=${perfcake.thread.count:100}"/>
            <property name="testExecutionName" value="Response time of
                ${server.host}"/>
         </destination>
      </reporter>
   </reporting>
</scenario>
```



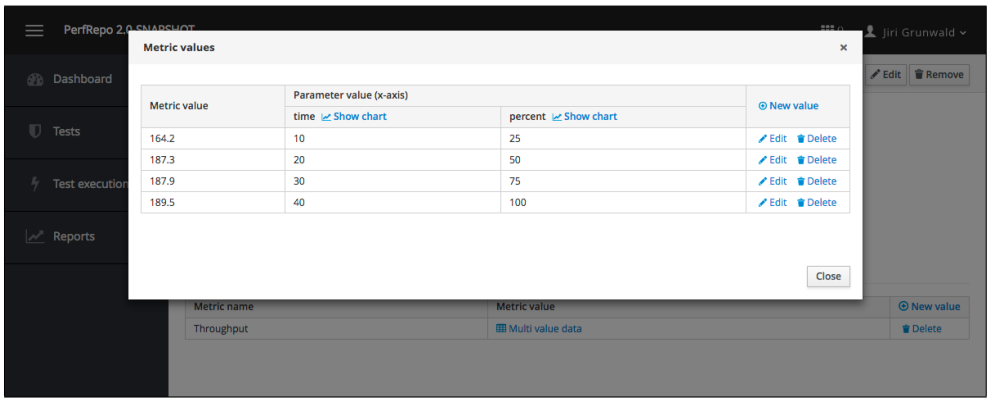
Obrázek 7.1: Snímek stránky s definicí testu



Obrázek 7.2: Formulář pro vytvoření podmínky pro metriku



Obrázek 7.3: Stránka s detailem běhu testu



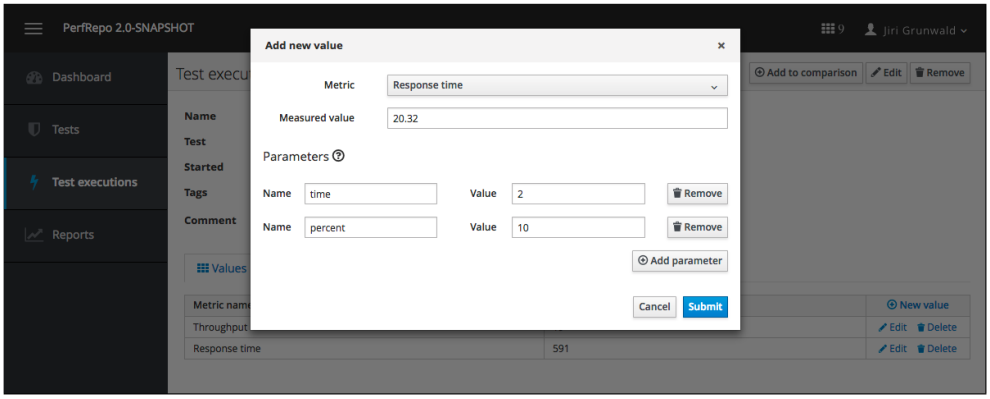
Obrázek 7.4: Zobrazení sekvence hodnot metriky v detailu běhu testu

pak je v tabulce zobrazen odkaz „Multi value data“, který zobrazí modální okno s hodnotami.

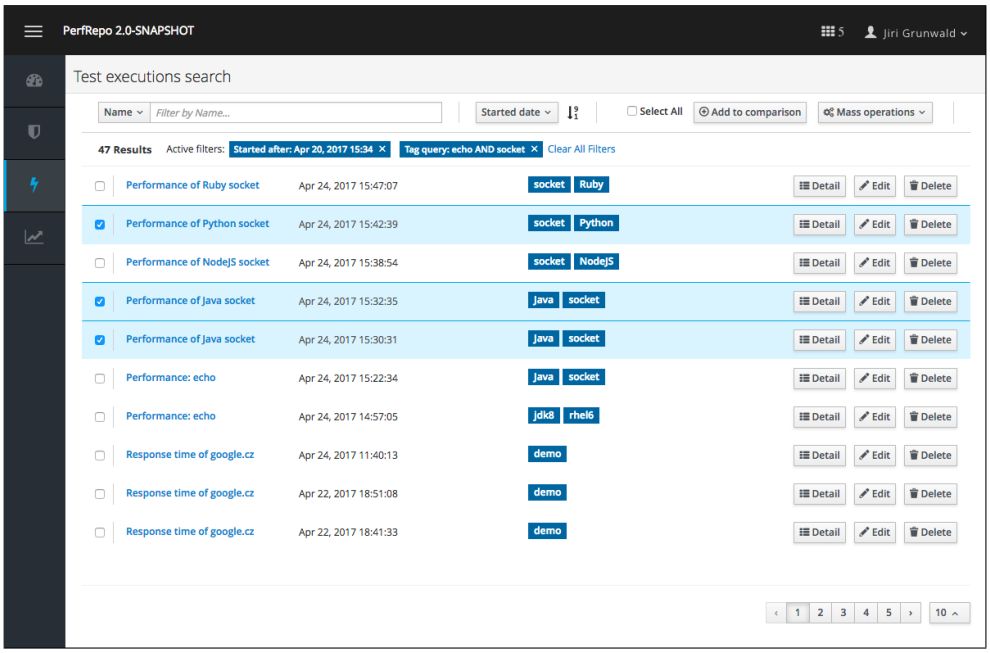
Ukázka zobrazení hodnoty metriky, která obsahuje sekvenci hodnot, je na snímku 7.4. Kliknutím na „Show chart“ se zobrazí naměřená data reprezentovaná ve spojnicovém grafu.

7.2.2.1 Zadání hodnoty metriky

Na snímku 7.5 je formulář vyvolaný z detailu běhu testu pro zadání hodnoty metriky. Důvod a popis pro zadávání parametrů hodnoty je popsán nápovědou („tooltip“).



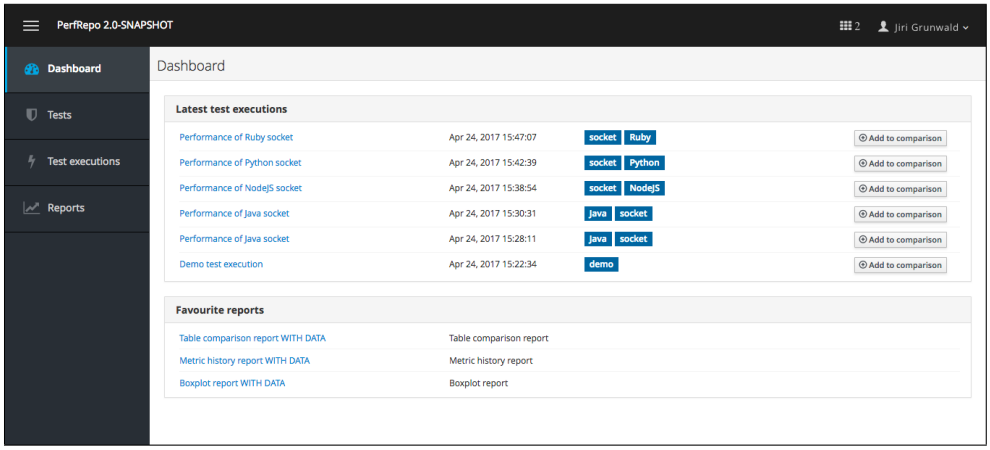
Obrázek 7.5: Formulář pro zadání hodnoty metriky



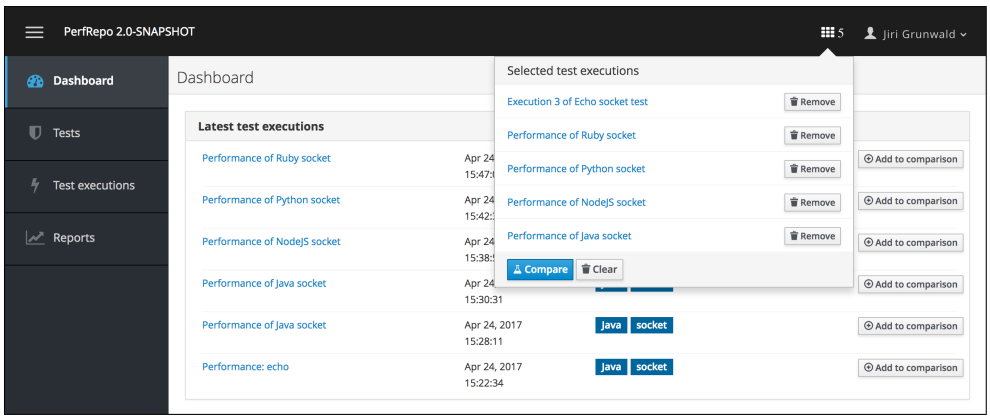
Obrázek 7.6: Stránka s vyhledáváním běhů testů

7.2.3 Vyhledávání běhů testů

Na snímku 7.6 je stránka s vyhledáváním běhů testů. Na snímku jsou vybrány tři běhy testu, se kterými můžeme provést hromadnou operaci (např. přidání štítku, přidání do seznamu k porovnání). Na snímku je také ukázka zúženého zobrazení menu aplikace.



Obrázek 7.7: Dashboard stránka s daty



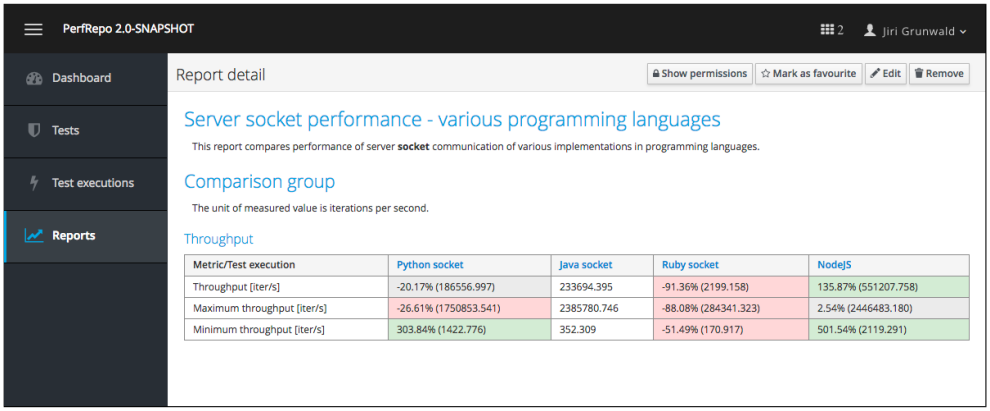
Obrázek 7.8: Menu s výběrem běhů testů k porovnání

7.2.4 Dashbord stránka

Na dashboard stránce je zobrazen panel s posledními běhy testů a panel s oblíbenými reporty. Ukázka stránky je na snímku 7.7. Názvy položek jsou odkazy na stránku s detailem položky. Dalším vylepšováním aplikace by se mělo využití dashboard stránky rozšířit a byla by vhodná i úprava stránky dle požadavků uživatele.

7.2.5 Menu s výběrem běhů testů k porovnání

Kliknutím na ikonku s číslem v pravé části hlavičky stránky se vyvolá menu s vybranými běhy testů k porovnání. Kliknutím na tlačítko „Compare“ se zobrazí náhled porovnání, ten lze poté uložit jako plnohodnotný report. Ukázka je na snímku 7.8.



Obrázek 7.9: Report porovnávající hodnoty vykonaných testů

7.2.6 Report porovnávající hodnoty vykonaných testů

Na snímku 7.9 je report, který porovnává hodnoty vykonaných testů. Z vybraných běhu testů se porovnají hodnoty společných metrik. V aplikaci se report jmenuje „Table comparison report“.

V případě, že metrika obsahuje sekvenci hodnot, je v buňce tabulky zobrazen odkaz „Compare to baseline“, případně „Compare all“ pro buňku referenční hodnoty. Po kliknutí se zobrazí modální okno, které obsahuje graf porovnání (spojnicový graf). Ukázka je na snímku 7.10.

7.2.7 Report pro průběžné sledování hodnoty metriky testu

V aplikaci se report jmenuje „Metric history report“. Měnící se hodnota metriky testu je v reportu reprezentována spojnicovým grafem. Každý bod grafu představuje jeden běh testu. Při kliknutí na bod, se zobrazí modální okno s informacemi o běhu testu, ukázka je na snímku 7.11.

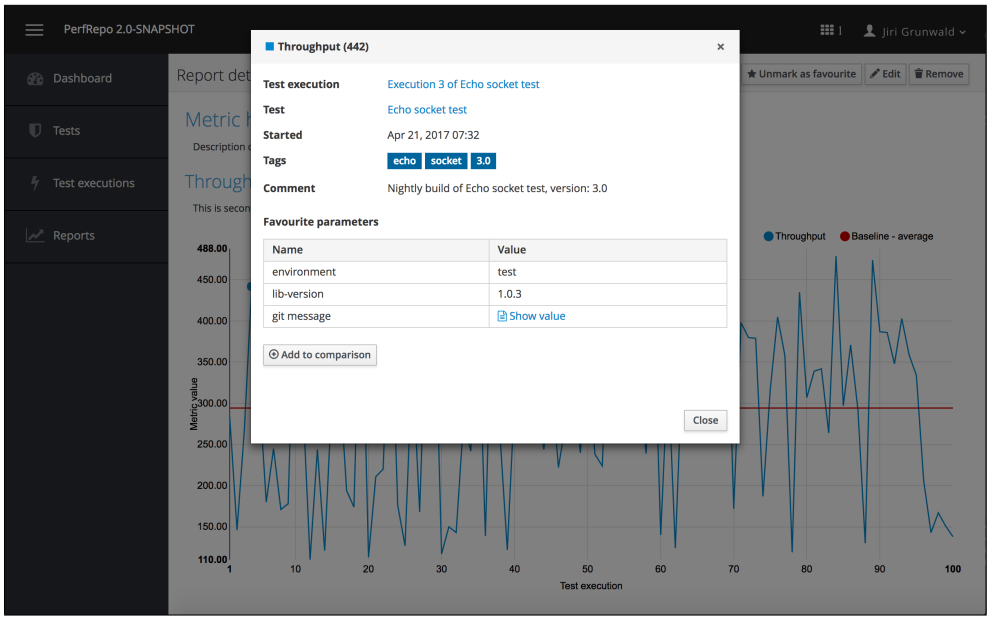
7.2.8 Report pro průběžné sledování metriky obsahující sekvenci hodnot

Na snímku 7.12 je ukázka reportu pro průběžné sledování metriky obsahující sekvenci hodnot. V aplikaci se report jmenuje „Boxplot report“. V tomto případě je porovnáváno pět běhů testů. Po najetí nad položku grafu se zobrazí legenda s hodnotami kvartilů.

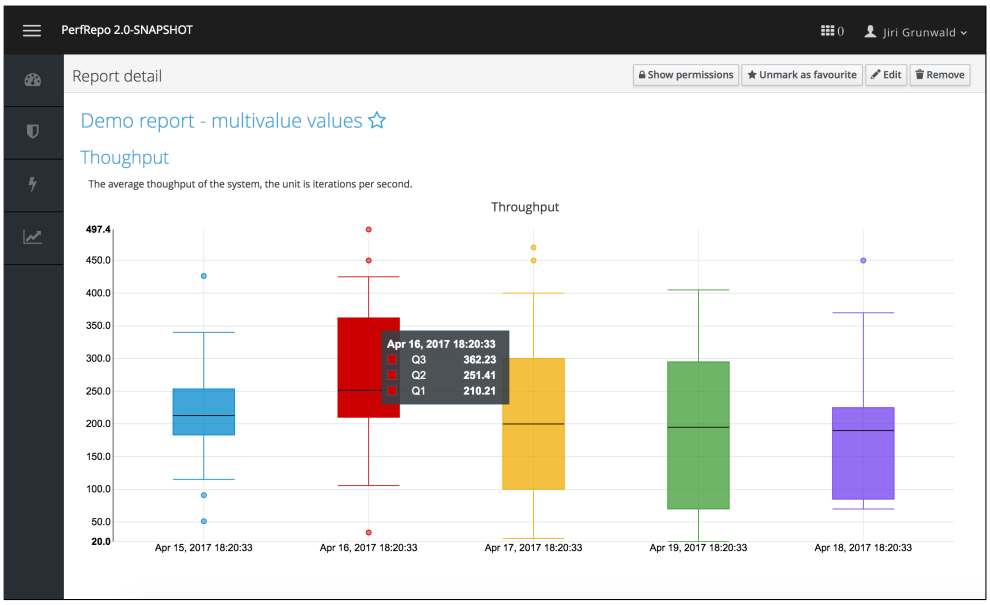
7.2.9 Report wizard

Průvodce pro vytvoření reportu má čtyři kroky. Na snímku 7.13 je třetí krok s konfigurací reportu pro průběžné sledování hodnoty metriky.

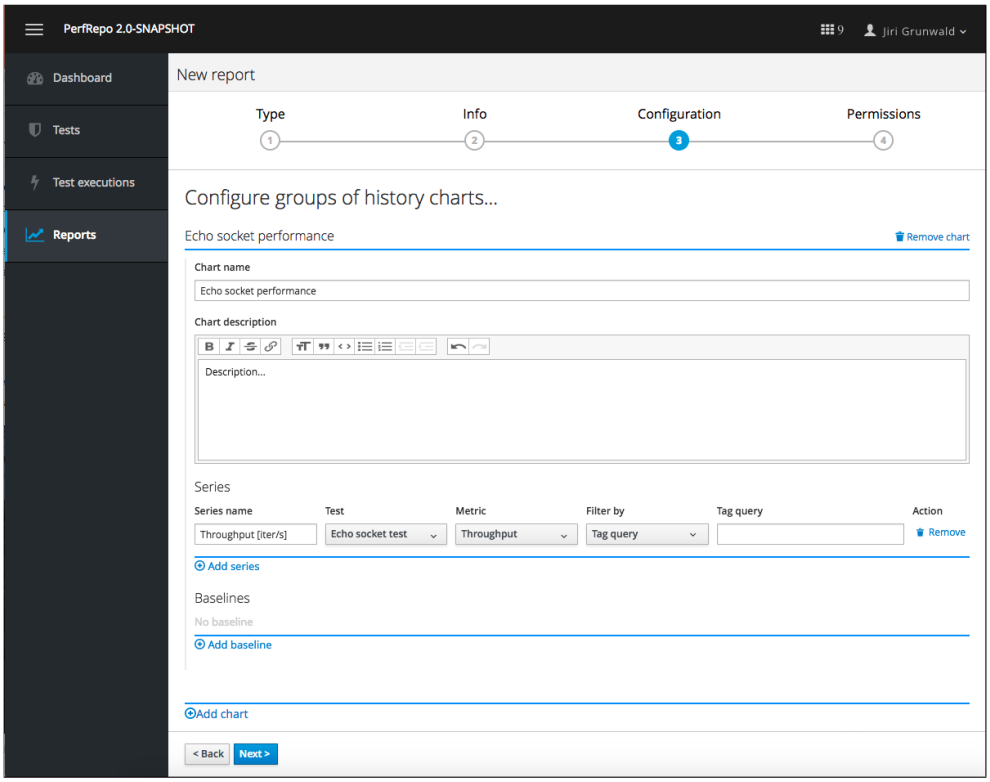
Obrázek 7.10: orovnání běhů testů s metrikou obsahující sekvenci hodnot



Obrázek 7.11: Report pro sledování hodnoty metriky



Obrázek 7.12: Report pro sledování metriky obsahující sekvenci hodnot



Obrázek 7.13: Konfigurace reportu pro průběžné sledování hodnoty metriky

Závěr

Na základě požadavků a analýzy současné verze aplikace PerfRepo 1.x bylo navrženo a implementováno nové uživatelské rozhraní. Změny si vyžádaly i kompletní přepsání webových služeb, což bylo i jedním z bodů zadání. Webové služby volají nově navržené rozhraní servisní vrstvy a aktuálně (konec dubna 2017) probíhá implementace tohoto rozhraní dalšími členy komunity. Funkcionalita byla testována pomocí dočasné implementace servisní vrstvy, která ověřuje funkčnost jednotlivých prvků uživatelského rozhraní.

Přepracováním bylo docíleno zvýšení použitelnosti aplikace, zjednodušila se možnost porovnat jednotlivé běhy testů, zlepšily se možnosti vyhledávání a práce s vyfiltrovanými položkami. Tvorba reportů byla přepracována do přehlednějšího průvodce. Dále přibyla například dashboard stránka.

Použití UI frameworku PatternFly se ukázalo jako dobrá volba, požadovanou funkcionalitu aplikace se podařilo tímto frameworkem docílit (především dobré možnosti filtrování obsahu). Vývoj byl ale lehce ztížen knihovnou AngularJS komponent pro PatternFly, která v sobě obsahuje pár chyb a místy neaktuální dokumentaci. Z implementace uživatelského rozhraní je potřeba ještě dokončit našeptávání štítků a parametrů běhů testu (dotaz může obsahovat logický součet a součin).

Demonstrace propojení a nahrání dat z nástroje PerfCake do aplikace PerfRepo bylo provedeno pomocí pluginu, ten byl pro účely práce přepsán, aby využíval nové REST API aplikace.

Aktuálním cílem je doladit novou verzi aplikace PerfRepo 2.0 pro finální vydání. Datum vydání je plánován na polovinu června 2017, vše ale bude záviset na postupu práce s implementací serverové části. V dalších verzích je plánována implementace uživatelských rolí a možnost vytvořit nového uživatele přímo v uživatelském rozhraní aplikace. Nových nápadů pro rozšíření

aplikace je více. Vhodné by bylo obohatit report pro porovnání běhů testů, který je teď reprezentovaný tabulkou, o sloupcový graf. Stránka s přehledem (dashboard) by mohla být rozšířena, aby umožňovala více možností zobrazení obsahu a podporovala úpravy dle požadavků uživatele.

Literatura

[1] Porter, A.: Regression Testing. [cit. 2017-03-23]. Dostupné z: https://www.cs.umd.edu/~aporter/html/currTesting.html

[2] SmartBear: How to Analyze Results of Your Load Tests. [cit. 2017-03-23]. Dostupné z: https://smartbear.com/learn/performance-testing/analyzing-results-of-load-tests/

[3] TestAutomationGuru: JMeter – Real Time Results – InfluxDB & Grafana. [cit. 2017-03-25]. Dostupné z: http://www.testautomationguru.com/jmeter-real-time-results-influxdb-grafana/

[4] Jenkins Wiki: Performance Plugin. [cit. 2017-03-24]. Dostupné z: https://wiki.jenkins-ci.org/display/JENKINS/Performance+Plugin

[5] TestAutomationGuru: JMeter Jenkins Integration. [cit. 2017-03-24]. Dostupné z: http://www.testautomationguru.com/jmeter-continuous-performance-testing-part2/

[6] Apache Software Foundation: Apache JMeter. [cit. 2017-03-23]. Dostupné z: http://jmeter.apache.org/

[7] Andrey Pohilko: Merge Results documentation :: JMeter-Plugins.org. [cit. 2017-03-23]. Dostupné z: https://jmeter-plugins.org/wiki/MergeResults/

[8] Apache Software Foundation: BackendListenerClient (Apache JMeter API). [cit. 2017-03-23]. Dostupné z: http://jmeter.apache.org/api/org/apache/jmeter/visualizers/backend/BackendListenerClient.html

[9] BlazeMeter: JMeter and Performance Testing for DevOps. [cit. 2017-03-23]. Dostupné z: https://www.blazemeter.com/

[10] BlazeMeter: Compare Reports. [cit. 2017-03-23]. Dostupné z: https://guide.blazemeter.com/hc/en-us/articles/207421555-Compare-Reports-Compare-Reports

[11] Gatling Corp: Gatling Load and Performance testing - Open-source load and performance testing. [cit. 2017-03-23]. Dostupné z: http://gatling.io/

[12] ThoughtWorks: Gatling: Take Your Performance Tests to the next Level. [cit. 2017-03-23]. Dostupné z: https://www.thoughtworks.com/insights/blog/gatling-take-your-performance-tests-next-level

[13] OctoPerf: JMeter VS Gatling Tool. [cit. 2017-03-23]. Dostupné z: https://octoperf.com/blog/2015/06/08/jmeter-vs-gatling/

[14] Locust Community: Locust – A modern load testing framework. [cit. 2017-03-23]. Dostupné z: http://locust.io/

[15] Qingshan Zhuan: Performance testing tools comparison between JMeter and Locust. [cit. 2017-03-23]. Dostupné z: http://qszhuan.github.io/test/2013/07/29/Comparison_between_JMeter_and_Locust

[16] Macík, P.; Večeřa, M.: PerfCake Plugins – User's Guide. [cit. 2017-03-23]. Dostupné z: https://www.perfcake.org/docs/user-guide/ug.extending.plugins.html

[17] PerfCake Community: JBoss Issue Tracker – PERFREPO. 2017, [cit. 2017-03-14]. Dostupné z: https://issues.jboss.org/projects/PERFREPO

[18] Pascal Precht: Two-way Data Binding in Angular. Dostupné z: https://blog.thoughtram.io/angular/2016/10/13/two-way-data-binding-in-angular-2.html

[19] Angular University: AngularJs vs Angular 2 – An In-Depth Comparison. [cit. 2017-03-19]. Dostupné z: http://blog.angular-university.io/introduction-to-angular2-the-main-goals/

[20] Tilde Inc.: Ember.js - Glossary: Web Development. 2017, [cit. 2017-03-14]. Dostupné z: https://guides.emberjs.com/v2.11.0/glossary/web-development/

[21] Sebastián Peyrott: More Benchmarks: Virtual DOM vs Angular 1 & 2 vs Others. [cit. 2017-03-20]. Dostupné z: https://auth0.com/blog/updated-and-improved-more-benchmarks-virtual-dom-vs-angular-12-vs-mithril-js-vs-the-rest/

[22] Paul Yoder's Blog: Why I recommend EmberJS over AngularJS. [cit. 2017-03-16]. Dostupné z: http://blog.yodersolutions.com/why-i-recommend-emberjs-over-angularjs/

[23] PatternFly Community: PatternFly | open interface project. [cit. 2017-03-16]. Dostupné z: https://www.patternfly.org/

[24] Red Hat, Inc.: OpenShift: PaaS by Red Hat, Built on Docker and Kubernetes. [cit. 2017-04-27]. Dostupné z: https://www.openshift.com/

[25] ZURB Inc.: Foundation | The most advanced responsive front-end framework in the world. [cit. 2017-03-16]. Dostupné z: http://foundation.zurb.com/

[26] Google, Inc.: Material design - Material design guidelines. [cit. 2017-03-16]. Dostupné z: https://material.io/guidelines/

[27] Google, Inc.: Material Design Lite. [cit. 2017-03-16]. Dostupné z: https://getmdl.io/

[28] Materialize: Documentation - Materialize. [cit. 2017-03-16]. Dostupné z: http://materializecss.com/

[29] Markov, D.: The 15 Best Material Design Frameworks and Libraries. [cit. 2017-03-17]. Dostupné z: http://tutorialzine.com/2016/03/the-15-best-material-design-frameworks-and-libraries/

[30] MockFlow: MockFlow - Online Wireframe and UX Tools. [cit. 2017-03-20]. Dostupné z: https://www.mockflow.com/

[31] NPM, Inc.: NPM. [cit. 2017-04-27]. Dostupné z: https://www.npmjs.com/

[32] Node.js Foundation: Node.js. [cit. 2017-04-27]. Dostupné z: https://nodejs.org/

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

CI Continuous Integration

CLI Command Line Interface

CSV Comma Separated Values

DOM Document Object Model

DSL Domain Specific Language

DTO Data Transfer Object

GUI Graphical User Interface

IDE Integrated Development Environment

JAR Java Archive

JMS Java Message Service

ORM Object Relational Mapping

POJO Plain Old Java Objects

QA Quality Assurance

REST Representational State Transfer

SASS Syntactically Awesome Style Sheets

SPA Single Page Application

TSDB Time Series Database

URL Uniform Resource Locator

UTC Coordinated Universal Time

WAR Web Application Archive

XML Extensible Markup Language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... popis obsahu CD
- `perfrepo-web.war` ..... WAR archív zkompilované aplikace
- `standalone.xml` ..... konfigurace aplikačního serveru WildFly
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Grunwald_Jiri_2017.pdf` ..... text práce ve formátu PDF
- `screens` ..... snímky obrazovky aplikace