ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Systém pro zpracování výstupů z frameworku Cucumber
Student:	Bc. Václav Štengl
Vedoucí:	Ing. David Buchtela, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem práce je návrh a implementace nástroje pro odlaďování testů, který umožní poskytnout vhodné uživatelsky přizpůsobené výstupy z frameworku Cucumber a to jak pro testery aplikací, tak pro vedoucí projektů.

1. Formou rešerše se seznamte s oblastí testování frontendových částí aplikací s využitím frameworku Cucumber.
2. Popište a zhodnoťte existující řešení pro reporting a statistiky testování pro framework Cucumber.
3. Navrhněte a implementujte systém, který umožní
   a) shromažďovat data z testů napsaných v jazyce Java a spuštěných přes Cucumber runner,
   b) odesílání dat na RESTful API,
   c) zobrazování uživatelsky přizpůsobených výstupů pro testery a vedoucí projektů.
4. Systém řádně otestujte.
5. Zhodnoťte náklady na vytvoření systému a odhadněte přínosy aplikace při řízení vývoje software.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 14. prosince 2016

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

Diplomová práce

Systém pro zpracování výstupů z frameworku Cucumber

Bc. Václav Štengl

Vedoucí práce: Ing. David Buchtela, Ph.D.

5. května 2017

Poděkování

Rád bych poděkoval Ing. Davidu Buchtelovi, Ph.D. za to, že byl ochotný stát se vedoucím této práce. Dále bych chtěl poděkovat kolegům Davidu Passlerovi za spolupráci na projektu, Janu Bacílkovi za trpělivost při code rewiev zdrojového kódu, Michalu Vojtíškovi za přidělování práce na jednotlivé sprinty a Zděnkovi Černému za to, že umožnil vznik tohoto projektu.

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 5. května 2017 .....

Odkaz na tuto práci

Štengl, Václav. *Systém pro zpracování výstupů z frameworku Cucumber.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato diplomová práce se zabývá vytvořením open source reportovacího nástroje pro testy napsané pomocí testovacího frameworku Cucumber [1]. Reportovací nástroj formou webové aplikace poskytne uživatelům několik různých pohledů na výsledky testů. První pohled poskytne technicky zdatnému uživateli detailní pohled na strukturu a výsledky testů, usnadní tak dohledávání chyb a údžbu testů v dobrém stavu. Druhý pohled poskytne manažerům shrnutí výsledků formou statistik a osobám interesovaným v průběhu implementace funkcionality přehled o postupu implementace pomocí pokrytí testy.

Práce obsahuje kapitoly, které popisují vývojový cyklus aplikace. Práce začíná kapitolou, která čtenáři přiblíží Cucumber framework, aby čtenář získal potřebné údaje o fungování frameworku a stávající řešení pro reportovaní. Dále je popsán proces vývoje mimo jiné obsahující sběr požadavků a jejich analýzu, návrh aplikace, její následnou implementaci a pokrytí implementace testy. Výsledkem bude funkční systém, který by měl nahradit stávající řešení reportování výsledků získaných spuštěním testů napsaných v Cucumber frameworku pro projekt, na kterém dlouhodobě pracuji. Po dohodě s projektovým manažerem budou zdrojové kódy dostupné na githubu [2] jako open source řešení. To platí i pro budoucí rozšiřování systému. Práce dále obsahuje podpůrné kapitoly s ekonomicko manažerským zhodnocením a postupem instalace.

Klíčová slova reportovací nástroj, Cucumber framework reportovací nástroj, open source reportovací nástroj, Cucumber reporty

Abstract

The presented thesis focuses on the development of an open source reporting tool for tests written using Cucumber framework. Firstly, theoretical chapters are covered, where Cucumber framework [1] is introduced in full detail and followed by the description of system requirements and detailed analysis. The evaluation of contemporary solutions has been performed regarding specified requirements and a final decision about the reporting solution has been made. Next, technical chapters are introduced. Those cover the whole SW development cycle, i.e. analysis, application design, and system implementation. As the final solution, the system was designed as a stand-alone web application, which will provide detailed report information for testers and thorough overview for project managers or board members. The thesis also covers supporting appendix chapters, such as an overview of the benefits provided by the system and an installation guide. The source code is going to be placed in a github [2] repository.

Keywords Cucumber framework reporting, Cucumber reporting tool, open source Cucumber reporting tool, Cucumber reporting

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Testování je neodmyslitelnou součástí vývoje, údržby a rozvoje aplikací. Dobře napsané testy pomáhají odhalit chyby v aplikacích prakticky ihned po jejich vzniku. Často však bývá problém testy vytvořit a udržovat podle platného zadání. Jedním z častých problémů je přesně popsat test pro problémovou doménu. Situace velmi často bývá taková, že osoby znalé promblémové domény nemají potřebné technické znalosti pro vytvoření a údržbu testů a vlastně to není ani náplň jejich práce. Naopak programátor/tester má potřebné technické znalosti, avšak nedostatečné znalosti problémové domény.

Tento problém se snaží řešit Cucumber framework, který rozděluje psaní testů na dvě vrstvy. První vrstvu tvoří feature soubory, které pomocí několika klíčových slov a jednoduchých vět v mluveném jazyce dovolují technicky neznalému člověku popsat testovací scénáře včetně vícenásobných průchodů s různými testovacími daty. Pro popsání testovacího scénáře uživateli stačí jednoduchý textový editor, tutoriál Cucumber frameworku, nebo si přečíst kapitolu o Cucumber frameworku v této práci. Vytvořený feature soubor se pak stane podkladem pro technicky zdatnou osobu k implementaci testů již bez potřeby zásahů od autora feature souborů. Výsledky spouštění testů je potřeba prezentovat ve vhodné podobě širokému spektru interesovaných osob.

Tato diplomová práce se tak zabývá vytvořením systému pro reportování výsledků testů napsaných v Cucumber frameworku pro osoby technicky zaměřené, které mají na starost stav testů, avšak i pro osoby, které zajímají testy z pohledu manažerského.

Pro vytvoření takového systému bylo nutné projít celý vývojový proces SW. Na počátku bylo nutné shromáždit požadavky zainteresovaných osob, přiřadit priority, rozpracovat detailněji analýzu požadavků pro jednotlivé fáze. Implementaci bylo kvůli rozsahu požadavků nutné rozdělit na několik fází, přičemž tato práce pokrývá první implementační fáze. Posléze započala implementace požadavků podle stanovených priorit pro první fázi. Implementace probíhala s využitím agilní metodiky. Po několika iteracích proběhlo demo pro zainteresované strany a na základě připomínek docházelo k změnám či

rozšíření požadavků a prioritizaci požadavků. Tímto způsobem proběhl vývoj aplikace až do podoby, ve které je dnes.

Kapitola 1

Cíl práce

Hlavním cílem práce je najít řešení, které by dokázalo shromažďovat data poskytovaná API Cucumber frameworku při spouštění testů napsaných a spoustěných pomocí testovacího frameworku Cucumber pro jazyk Java. Tyto data aplikace musí následně využívat pro vytvoření reporů a statistik pomocí webové aplikace.

K naplnění tohoto cíle bylo potřeba splňit řadu dílčích cílů. Nejdříve bylo třeba prostudovat existující řešení a pokusit se pomocí nich splňit požadavky specifikované v kapitole požadavky a analýza. Jak se následně můžete dočíst v kapitole existující řešení z různých důvodů nesplňila očekávání a tak bylo potřeba stanovit další cíl a tím je takové řešení vytvořit. Nezdálo se však vhodné upravit stávající řešení a tak jsem se rozhodl pro implementaci systému na zelené louce.

Řešení je pojato jako systém, který se bude skládat ze tří hlavních částí. Kde první část bude určena výhradně pro sběr dat z testů spustěných pomocí Cucumber frameworku. Druhá část bude tyto výsledky přijímat pomocí REST-API a zajistí jejich validace a persistenci. Třetí část systému pak slouží k zobrazování nashromážděných výsledků ve formě reportů a statistik. K dosažení tohoto cíle je nutné překonat mnoho překážek a splňit mnoho dílčích úkolů.

Nejhrubější dělení dílčích úkolů je následující. Ze shromážděných požadavků na systém je nutné vytvořit detailnější analýzu podle níž bude možné navrhnout architekturu systému, určit potřebné technologie k vývoji a implementovat požadovanou funkcionalitu. Implementaci je následně nutné pokrýt testy. Po otestování aplikace může započít integrace se systémem zajišťujícím CI a CD v této práci takovým systémem bude Jenkins [3]. Po úspěšné integraci bude následovat postupný přechod ze stávajícího řešení na nové a započne testovací provoz s potřebnou aplikační podporou a bug fixingem.

Reportovací nástroj je pojat jako open source projekt a bude ke stažení v gitovém repozitáři stengvac/cucumber-reports [4].

Kapitola 2

Cucumber framework

Od roku 2008 Cucumber framework [1] roste a každým rokem získává na popularitě. Framework vznikl pro jazyk Ruby [5], ale dnes je možné psát testy s pomocí Cucumber frameworku ve více než 10ti programovacích jazycích viz Cucumber implementations [6]. Pro tuto práci je důležité, že Java [7] je jedním z nich. Jedná se o open source testovací nástroj, který napomáhá ve psaní testů. Pomocí Cucumber frameworku je možné psát automatické testy v behavioral driven development (BDD) stylu. BDD vychází z metodologie pro vývoj SW test driven development, kde pro každý kousek kódu by měl vývojář vytvořit:

- jednotkové testy,
- testy by měly skončit chybou,
- implementovat funkcionalitu,
- znovu testy spustit a ověřit, že prošly.

S tímto přístupem by si měl vývojář nejdříve uvědomit co vlastně programuje a pak vytvořit implementaci. TDD k tomuto přístupu přidává chování v problematické doméně. Tedy BDD se snaží k testům přidat význam z pohledu byznysu. Testy se tak řídí podle příběhů zajímavých pro byznys popsaných pomocí terminologie domény.

Framework dělí psaní testů do dvou vrstev. Jak se v následující kapitole pokusím vysvětlit, jedná se o obrovskou výhodu pro testování aplikací. Při testování implementované funkcionality nastává velmi často problém s definicí testů z pohledu uživatele, případně osoby znalé problémové domény a technické části testu. Tento problém Cucumber framework řeší již zmíněným rozdělením testů na dvě vrstvy.

2.1 Gherkin

První vrstva testů je psaná pomocí jazyka Gherkin [8]. Jedná se o jazyk, který připomíná psané slovo. Tomuto jazyku rozumí parser Cucumber frameworku i osoby znalé problematické domény. Jedná se o dříve zmíněné příběhy, které popisují testovací scénáře. Zápis v takové formě umožňuje osobám znalým problematické domény porozumět a v případě potřeby i napsat či upravit testovací scénáře. Jazyk tak slouží pro hned několika účelům.

- Je popisný a lze napsané testy použít jako dokumentaci funkčnosti.
- Specifikuje průchody pro automatické testování.
- Nabádá programátory/testery jakou funkcionalitu umístit do jednotlivých metod. Jelikož jsou jednotlivé kroky popsány formou jednoduchých vět, které popisují co se má stát při průchodu kroku.

Při použití jazyka gherkin je však nutné dodržovat určité konvence. Testovací scénáře se píší do zdrojových souborů. Každý zdrojový soubor obsahuje právě jednu testovanou feature. Feature do češtiny lze přeložit jako vlastnost systému, avšak budu se držet slova feature. Jednu feature lze otestovat několika testovacími scénáři. Druhé pravidlo je, že zdrojové soubory musí mít příponu `.feature`. Dále v práci budu takové soubory označovat jako feature soubory.

Pro představu, jak může takový feature soubor vypadat, přikládám příklad převzatý z [8]. Podprobnější popis feature souboru představím dále.

Code 2.1: Ukázka feature souboru

```
Feature: Some terse yet descriptive text of what is desired
  Textual description of the business value of this feature
  Business rules that govern the scope of the feature
  Any additional information that will
  make the feature easier to understand

  Scenario: Some determinable business situation
    Given some precondition
      And some other precondition
    When some action by the actor
      And some other action
      And yet another action
     Then some testable outcome is achieved
      And something else we can check happens too

  Scenario: A different situation
   ...
```

2.2 Podrobnější popis

Následující sekce se zabývá podrobnějším rozborem syntaxe a možného obsahu feature souborů. Tuto sekci zde uvádím, aby si čtenář udělal obrázek jak lze použít Gherkin a jaké informace lze do feature souborů zapsat.

2.2.1 Klíčová slova

Gherkin využívá pro zápis testů v feature souborech jen několik klíčových slov. Je poměrně snadné napsat první feature soubor již po přečtení tutoriálu na Cucumber stránkách. Feature soubory lze psát v různých jazycích Cucumber framework jich podporuje velké množství. Ale pokud se chystáte použít feature soubory pro automatizaci testů, pak bych doporučil angličtinu. To platí pro klíčová slova i obsah vět. Oboje lze lokalizovat do mnoha jazyků.

Započnu ukázkou jednoduchého feature souboru, který obsahuje skoro všechna klíčová slova. Poté se pustíme do rozboru významu jednotlivých klíčových slov. Příklad je převzat z [9] a mírně poupraven pro potřeby ukázky všech klíčových slov v jednom feature souboru. Řádky začínající symbolem # obsahují doprovodné komentáře k feature souboru.

Code 2.2: Ukázka feature souboru se všemi klíčovými slovy

```
#feature zabyvaji se prodejem kavy
Feature: Serve coffee
  #detailnejsi popis feature i vice radkovy
  Coffee should not be served until paid for
  Coffee should not be served until the button
  has been pressed
  If there is no coffee left then money should be refunded

  #spolecne 'predpoklady' pro vsechny scenare v feature
  Backkground
   Given A lot of customers were server today
     And Store is running low on ingredients

  #scenar popisujici nakup posleni kavy
  Scenario: Buy last coffee
   Given there are 1 coffees left in the machine
     And I have deposited 1$
    When I press the coffee button
    Then I should be served a coffee

  #parametrizovany scenar s vicero pruchody
  Scenario Outline: Buy product
   Given I have <money>$
```

```
    When I buy <product>
    Then I will obtain <product>

  #zdroj dat pro pruchody scenario outline sekce
  #data jsou dostupna pod jmenem def v prvnim radku
  Examples:
    |money|product|
    | 1   | orange|
    | 10  |  cake |
```

2.2.2 Klíčová slova a jejich význam

Gherkin pro popis feature souborů momentálně používá 12 klíčových slov v anglickém jazyce. Jazyk gherkin lze použít v různých jazykových mutacích a lze tedy psát feature soubory v různých jazycích. Podle použitého jazyka se může lišit i počet klíčových slov. Rozdíl nastává u klíčových slov použitých pro kroky. Při vysvětlování klíčových slov v následujících sekcích využiji anglická klíčová slova.

Feature Klíčové slovo feature je vždy na samém počátku zdrojového souboru a je možné ho použít v souboru pouze jednou. Deklaruje zastřešující testovanou vlastnost, pro kterou do feature souboru píšeme testovací scénáře. Ke klíčovému slovu můžeme připojit název vlastnosti. Jedná se o text na stejném řádku jako je uvedené klíčové slovo. Dále je možné připojit i detailnější popis vlastnosti. Tedy všechen text pod klíčovým slovem až k dalšímu klíčovému slovu, který není komentář.

Příklad:

Code 2.3: Ukázka použití klíčového slova Feature

```
Feature: Serve coffee
Coffee should not be served until paid for
Coffee should not be served until the button
has been pressed
```

První řádek je název vlastnosti a zbylé dva řádky jsou detailnější popis vlastnosti.

Background Deklaruje počátek sekce, která popisuje společné předpoklady, prostředí a výchozí stav pro všechny scénáře ve feature souboru. Kroky uvedené v Background sekci se provedou vždy před průchodem scénářem. Background sekce nemusí být ve feature souboru definována a feature soubor může obsahovat maximálně jednu Background sekci. Příklad: mějme feature pro testování určité části internetového bankovnictví (IB).

Do Background sekce můžeme například umístit kroky pro přihlášení do IB.

Příklad:

Code 2.4: Ukázka použití klíčového slova Background

```
Backkground
    Given I have username: user
      And I have password: passwd
      And I have logged into IB
```

Předchozí příklad uživatele před každým tetovacím scénářem přihlásí do IB.

Scenario Deklaruje počátek sekce pro popis scénáře. Scénář při automatizovém spuštění bude mít právě jeden průchod. Kroky ve scénáři musí mít již všechny potřebné informace obsažené v sobě. Kroky scénáře mohou následně být použity pro implementaci metod, které se postarají o automatický průchod scénářem. Feature může obsahovat neomezené množství testovacích scénářů.

Příklad:

Code 2.5: Ukázka použití klíčového slova Scenario

```
Scenario: Buy last coffee
    Given there are 1 coffees left in the machine
      And I have deposited 1$
     When I press the coffee button
     Then I should be served a coffee
```

Příklad popisuje scénář pro nákup poslední kávy. V krocích scénáře jsou už zachyceny všechny potřebné údaje.

Examples Sekce vnořená v sekci Scenario Outline, jejíž popis bude následovat. Poskytuje sekci Scenario Outline data pro opakované průchody scénářem. Data jsou uvedena formou tabulky a v Scennario Outline sekci jsou data dostupná pod názvem sloupce, tj. první řádek tabulky. Každý průchod Scenario Outline sekce pak pro každý svůj průchod dostane jeden řádek. Řádky se prochází postupně v pořadí v jakém jsou uvedeny.

Příklad:

Code 2.6: Ukázka použití klíčového slova Examples

```
Examples:
    |money|product|
    | 1   | orange|
    | 10  |  cake |
```

Data z předchozího příkladu je možné použít v Scenario Outline pod názvy **money** a **product**.

Scenario outlice Deklaruje počátek scénáře, který obsahuje kroky se zástupci. Zástupci jsou během opakovaného spouštění scénáře nahrazeny daty z Examples sekce. Každý průchod používá jeden řádek z Examples sekce. Počet průchodů scénáře je dán počtem řádků Examples sekce. V krocích je nutné použít znaky < a > pro označení zástupce.

Pozn: Data z Examples sekce se mohou ještě před použitím v zástupcích modifikovat. Příklad:

Code 2.7: Ukázka použití klíčového slova Scenario Outline

```
Scenario Outline: Buy product
        Given I have <money>$
         When I buy <product>
         Then I will obtain <product>

Examples:
    |money|product|
    | 1   | orange|
    | 10  |  cake |
```

Scenario Outline z příkladu proběhne 2 krát. Přičemž v prvním průchodu se využijí data **money = 1** a **product = orange**.

2.2.3 Klíčová slova pro kroky

Než přejdu k účelu jednotlivých klíčových slov určených pro kroky, je vhodné vysvětlit kroky jako takové. Krok v feature souboru je vlastně jednoduchá věta, která začíná jedním z klíčových slov. Věty by měly stručně popisovat co se má v daném kroku stát. Věty je vhodné psát krátké a jednoduché, jelikož jsou často podkladem pro implementaci automatizovaných testů. Klíčová slova nemají v podstatě žádný smysl z hlediska funkcionality a všechny se chovají stejně. Jejich účel je čistě dokumentační.

V anglické verzi existuje 5 klíčových slov pro deklaraci kroků. V jiných jazycích může klíčových slov existovat více, avšak stále jsou kombinací těchto 5 základních.

Používání klíčových slov není nikterak omezeno do množství ani pořadí.

Given Označuje kroky, které uvádí systém do známého stavu potřebného pro následné operace. Obvykle slouží pro přípravu dat, prostředí a přechodu do počátečního stavu.

When se používá pro provedení klíčových akcí při průběhu scénaře, které jsou pro nás v rámci scénáře důležité a zajímají nás. Jejich spuštěním chceme vyvolat akce, jejichž výsledky se dále v scénáři chystáme

hodnotit. Příkladem může být stisk tlačítka, které vyvolává související akce. Obvykle následuje krok s počátkem Then a hodnotíme výsledky akce.

Then slouží pro sledování nebo kontrolu výsledků předchozích kroků.

And, But Účel těchto klíčových slov je zpřehlednit psaní scénářů. Používají se místo opakování prvních tří klíčových slov. Pro lepší představu následuje příklad.

Možný popis testovacího scénáře pouze pomocí Given, When, Then. Příklady jsou převzaty z [10] společně s dalšími informacemi ohledně kroků.

Code 2.8: Ukázka popisu scénáře

```
Scenario: Multiple Givens
  Given one thing
  Given another thing
  Given yet another thing
   When I open my eyes
   Then I see something
   Then I don't see something else
```

Popis testovacího scénáře zpřehledněného pomocí klíčových slov **And** a **But**.

Code 2.9: Ukázka popisu scénáře s @And a @But

```
Scenario: Multiple Givens
  Given one thing
    And another thing
    And yet another thing
   When I open my eyes
   Then I see something
    But I don't see something else
```

V testech se lze dále setkat s parametrizovanými kroky pomocí zástupných řetězců, za které se při běhu testů dosazují data z Examples sekce. Zástupné řetězce se označují pomocí znaků < a >. Příkladem:

```
Scenario Outline:
Given I have <money>

Examples:
| money |
| 50    |
```

Se znalostí výše uvedených klíčových slov a pomocí vět v anglickém jazyce (lze použít ekvivalenty klíčových slov z dalších jazyků, ale doporučil bych držet

Tabulka 2.1: Česká a anglická klíčová slova pro kroky

Anglická klíčová slova	Česká klíčová slova
• Given	• Zapředpokladu
• When	• Když
• Then	• Pak
• And	• A
• But	• Ale
	• Pokud
	• Zapředpokladu

se angličtiny nebo alespoň vynechat diakritiku kvůli použití v programovacích jazycích) lze popsat testovací scénáře včetně použitých testovacích dat.

Pro představu jsem si dovolil uvést anglická a česká klíčová slova pro deklaraci kroků. Uvedená klíčová slova jsou zkopírované anotace pro jazyk Java z package a `cucumber.api.java.en` a `cucumber.api.java.cs`.

Poznámka: pro propojení kroků uvedených v feature souboru a programovacím jazykem jsou použity právě anotace (v případě jazyka Java) z package cucumber.api.java.<lang>. Pokud se rozhodnete psát feature soubor v jiném jazyce než anglickém, pak i anotace budou v tomto jazyce a v příkladu uvedeném výše je vidět, že klíčová slova pro kroky mohou obsahovat i diakritiku. Doporučoval bych tedy použít angličtinu minimálně pro klíčová slova.

2.3 Atributy v feature souboru

V předchozí sekci čtenář získal základní představu o klíčových slovech potřebných ke psaní testovacích scénářů. Nyní rozšířím čtenářovu znalost o povolené atributy u jednotlivých klíčových slov. Je to tak další možnost, jak přidat informace ke scénářům a tudíž je potřeba je přenést i do reportů.

Započnu vyjmenováním druhů atributů ať čtenář získá představu, co lze v feature souboru očekávat a jaký má atribut význam.

Tag je řetězec začínající znakem @ a musí předcházet klíčovým slovům Feature, Scenario, Scenario Outline, Examples. Tagy mají různé druhy využití.

- Lze s nimi řídit, kdy se má feature/scénář spustit. V příkladu se spouštěním feature souboru pomocí Java runneru v anotaci `cucumber.api.CucumberOptions` existuje atribut tags. Do atributu je možné definovat seznam tagů (je nutné uvést název tagu včetně @), které se mají v tomto runneru spouštět. To znamená, že pokud klíčové slovo v feature souboru neobsahuje jeden z potřeb-

ných tagů, pak se nespustí. Do seznamu lze definovat i negace a to pomocí znaku ~před tag.

- Lze pomocí nich přidávat informace o testovacím scénáři. Například: @HappyScenario, @LoginTest. @Bug,
- Lze pomocí nich vytvořit mapování na úkoly v ticketovacím nástroji a tím ulehčit hledání testů například při opravě bugu. Příklad: @MyProject1254
- Lze je využít při mapování na popis implementované funkcionality (vedené například v Enterprise Architectu [11] v a tím dokázat, že funkcionalita je již naimplementovaná a funkční.

Komentář je vše od symbolu # až do konce řádku. Na rozdíl od tagu lze komentář umístit prakticky kamkoliv do .feature souboru.

Name alias název sekce. Atribut name je text uvedený za některými klíčovými slovy a poskytuje popisnější informaci pro následný blok. Příkladem může být název scénáře.

Description jedná se o veškerý text, který není komentář a je umístěn v sekci klíčového slova až do dalšího klíčovéhlo slova.

DataTable tento atribut se vystytuje pouze u kroků. Deklaruje se stejně jako tabulka v Examples sekci. Slouží pro předání složitějších datových struktur z feature souboru do implementace metody jako parametr. V hlavičce metody se může objevit jako mapa hodnot nebo seznam objektů (je potřeba vytvořit expovídajícíc objekt s atributy jako jsou názvy sloupců v tabulce).

Znalost atributů je žádoucí, jelikož je potřeba propagovat je do reportů. O tom ale více v kapitole požadavky a analýza.

2.4 Výsledky

Předchozí sekce snad poskytly čtenáři představu, jaké informace lze získat ze samotného feature souboru. Pro úplnost ještě doplním, že je potřeba ukládat i data z běhu testů.

Asi nejdůležitější informace poskytovaná při běhu scénářů je výsledek běhu kroku. Od něj se pak určuje výsledek běhu scénáře, celého feature souboru i sestavení.

Výsledek kroku může mít několik různých hodnot. Nejsme omezeni jen na hodnoty kroku prošel/neprošel. Cucumber framework celkem definuje pět různých stavů.

PASSED krok proběhl v pořádku.

FAILED nastala chyba při vykonávání kroku.

UNDEFINED implementace kroku není v souboru specifikovaném atributem glue, který specifikuje kde hledat implementace kroků k nalezení.

PENDING definice kroku existuje, avšak není hotová implementace.

SKIPPED krok scénáře, který nastal po jednom z chybových kroků v scénáři.

Další poskytnuté údaje jsou doba trvání kroku a vyjímka, která způsobila selhání.

Průchody jednotlivých kroků mohou generovat embedované soubory nejčastěji obrázky.

2.5 Automatické testy

Se znalostí syntaxe již zbývá jen vytvořit automatický test. Následující příklad je pro maven [12] project. K vytvoření automatického testu potřebujeme celkem tři soubory a několik závislostí na Cucumber frameworku.

Nejdříve je nutné do pom.xml přidat potřebné závislosti. Jedná se o dvě závislosti na Cucumber framework a jednu pro jUnit [13].

Code 2.10: Maven závislosti pro spuštění Cucumber přes jUnit

```
<properties>
        <version.cucumber>1.2.5</version.cucumber>
        <version.junit>4.12</version.junit>
</properties>

<dependencies>
        <dependency>
                <groupId>info.cukes</groupId>
                <artifactId>cucumber-java</artifactId>
                <version>${version.cucumber}</version>
                <scope>test</scope>
        </dependency>
        <dependency>
                <groupId>info.cukes</groupId>
                <artifactId>cucumber-junit</artifactId>
                <version>${version.cucumber}</version>
                <scope>test</scope>
        </dependency>

        <dependency>
```

```
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${version.junit}</version>
            <scope>test</scope>
        </dependency>
</dependencies>
```

První potřebný soubor je feature soubor s popisem testované vlastnosti systému. V javovských projektech se feature soubory obvykle ukládají do test/resources/package/<featurename>.feature. Asi nejjednodušší možný feature soubor může vypadat takto.

Code 2.11: Minimalistický feature soubor

```
Feature:

Scenario:
Given Test step
```

Řekněme, že se jmenuje myfeature.feature a je umístěn dle výše uvedeného popisu v package `cz.cvut.fit`.

Druhý potřebný soubor je soubor, který v sobě bude obsahovat implementace kroků. V našem případě se jedná o Java třídu. Často se jako postfix pro pojmenování takových tříd používá řetězec Steps. Většina IDE již podporuje předgenerování metod a to i s potřebnou anotací podle existujícího feature souboru. Vygenerovaná anotace odpovídá použitému klíčovému slovu a její hodnota je věta použitá pro popis kroku. Hodnota může obsahovat regulární výrazy pro použité záslepky. Cucumber framework pak při běhu scénářů namapuje kroky v feature souboru na implementaci právě dle této hodnoty a jako parametry metody dosadí řetězce, které jsou součástí hodnoty a tvoří reálné hodnoty záslepek.

Já osobně využívám IntelliJ IDEA [14], která předgenerování metod zvládá. Po stisku klávesové zkratky Alt+Enter. Pozor musíte kurzorem stát na definici kroku v feature souboru. Se zobrazí kontektové menu, do kterého je nutné zadat třídu, kam IDE vygeneruje metody s anotacemi. Dobrým zvykem je třídu umístit do stejného package, ve kterém je umístěný feature soubor a přidat subpackage steps. V našem případě tak výsledný package má tvar cz.cvut.fit.steps a jméno třídy například MyFeatureSteps.java.

Výsledek by měl být podobný tomuto.

Code 2.12: Implementace kroků pro feature soubor

```
import cucumber.api.PendingException;
import cucumber.api.java.en.Given;

public class MyFeatureSteps {
```

```
        @Given("^Test step$")
        public void testStep() throws Throwable {
        // Write code here that turns the phrase
        //above into concrete actions
        throw new PendingException();
        }
}
```

Pro účely tutoriálu není potřeba třídu upravovat. Při spuštění testu scénář selže kvůli vyjímce, ale to signalizuje, že se spuštění povedlo.

Poslední soubor je pro spouštění testů skrze jUnitový runner. Opět se jedná o Java třídu s několika anotacemi, které řídí spouštění testů.

Vytvořte soubor MyFeatureRunner.java v package cz.cvut.fit. Pro spuštění testu bude nutné použít dvě anotace. První z nich je anotace `org.junit.runner.RunWith`, která se používá pro spuštění testu se třídou, která je vypoužita jako hodnota anotace v tomto případě se jedná o třídu `cucumber.api.junit.Cucumber`, která se stará o spouštění feature souborů s nastavením v druhé anotaci. Druhá anotace `cucumber.api.CucumberOptions` poskytuje při spuštění testů pomocí Cucumber runneru potřebné informace pro běh pomocí atributů anotace. Atribut features specifikuje, které feature soubory se mají spustit a druhý atribut glue určuje kde má Cucumber runner hledat implementace kroků.

Code 2.13: Třída pro spuštění feature souboru

```
import org.junit.runner.RunWith;

import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = {
        "classpath:cz.cvut.fit/myfeature.feature"
        },
        glue = "cz.cvut.fit.steps")
public class MyFeatureTest {
}
```

Po spuštění tohoto souboru pomocí IDE nebo příkazem `mvn clean install` se spustí i námi naspaný test a pokud se všechno povedlo, měl by skončit neúspěchem kvůli Pending vyjímce.

2.6 Využití cucumber frameworku

Pojdme si nejdříve připomenout něco málo o testování aplikací. V následující části se pokusím čtenáři přiblížit testování zejména rozsáhlých aplikací. U rozsáhlých projektů, které jsou ve vývoji i několik let, je nutné pokrýt zdrojový kód kvalitními testy a dělit testy do několika úrovní. Kvalitní testy pomohou odhalit chyby v kódu velmi brzo po jejich vzniku a dělení na úrovně zajistí přehlednost a omezení oblasti, kde se může chyba nacházet. Asi nejznámější rozdělení testů je dle [15] následné:

Testování programátorem (Developer testing): Jedná se o test kódu nebo jeho prověření po logické stránce dalším programátorem „někdy také nazývané jako test čtyř očí“ [15].

Často tímto způsobem lze odhalit a odstranit velké množství chyb, špatných praktik a nešvarů ještě před tím, než se nově napsaný kód dostane do centrálního repozitáře. Často využívanou technikou tak může být code review. Jedná se o techniku, kdy je nově napsaný zdrojový kód ještě před přidáním do centrálního repozitáře odchycen a musí projít několika kontrolami.

První kontrola je automatická, kdy se kód v centrálním repozitáři včetně nového přírůstku nechá sestavit pomocí nástroje pro Continuos Integratio and Continous Delivery (například Jenkins). Při sestavení je možné spouštět i statickou analýzu kódu a pokud kód obsahuje příliš mnoho přestupků nebo přestupky s přílišnou závažností oproti deklarovaným pravidlům, tak lze kód zastavit, dokud programátor přestupky neopraví.

Další krok je kontrola kódu dalším programátorem. Z vlastní zkušenosti mohu doporučit Gerrit [16]. Jedná se o webovou aplikaci, která umožnuje procházet přírůstky kódu, diskutovat nad kódem a následně kód ohodnotit v rozmezí <-2, 2>. Kde -2 v podstatě znamená velmi špatný kód, který je potřeba kompletně přepsat. Ohodnocení -1 značí, že v kódu jsou místa která jsou potřeba upravit. Ohodnocení 0 se obvykle používá jen k propagaci kometářů. Ohodnocení +1 znamená, že je kód v podstatě v pořádku, ale kontrolu by měla provést další osoba. A ohodnocení +2 se používá, když druhý programátor uzná, že je kód připraven k přidání do centrálního repozitáře. Následuje odeslání kódu do centrálního repozitáře. Zavedení code review nemusí být zrovna snadné, ale výsledek za to stojí, navíc postupem času si programátoři zvyknou, že mají dodržovat určitá pravidla a celý proces se tím urychlí.

Testování jednotek (Unit testing): Testování jednotek je další důležitou úrovní v testování SW, jelikož zaručujeme správnost implementace od nejmenších možných kousků a tím značně omezujeme oblast, kde mohla nastat chyba. Unit testy jsou obvykle krátké a kromě simulace komunikace s okolím obvykle nevyžadují další prostředky. Jedná se o testování

zdrojového kódu na úrovni jednotlivých tříd. Základní myšlenka je otestovat pouze atomickou jednotku a tím zaručit její správné chování bez reálné interakce s okolím. Veškerá interakce s okolím je falšovaná - programátor v testu definuje jaké odpovědi od okolí dostane za definované vstupy.

Funkční testování: „Určeno pro ověření, že systém obsahuje funkcionalitu, která je požadována. Obecně vzato testují se všechny implementované funkce a jejich správná funkčnost z hlediska požadavků zákazníka.

Z předchozí definice lze usoudit, že tento druh testů je velmi důležitý pro výslednou bezporuchovost aplikace. Proto je na tento druh kladen velký význam během testovacího cyklu.“ [17]. Kromě ověření funkčnosti mohou být testy také důkazem o funkčnosti aplikace pro zákazníka.

Integrační testování: Jedná se o testování, při kterém se postupně testuje integrace částí uvnitř komponent. Například se na sebe integrují jednotlivé API v komponentě nebo integrace s HW potřebným k běhu aplikace. Následně probíhá integrace mezi jednotlivými komponentami v systému. Tyto testy již většinou nepíší programátoři, ale tým testerů.

Systémové testování: Po úspěšné integraci jednotlivých komponent pomocí integračních testů, je načase otestovat integraci aplikace jako celku hlavně v pozdějších fázích vývojového cyklu. Často slouží jako poslední kontrola SW před odevzdáním zákazníkovi. Na rozdíl od funkčních testů, kde se také testuje požadovaná funkcionalita samostatně se zde provádí testování komplexnějších scénářů. Projeví se tedy, zda se scénáře nějakým způsobem vzájemně neovlivňují.

UAT – Akceptační testování: Je testování, které se provádí na straně zákazníka a to i na jeho prostředí. Akceptační testování by mělo proběhnout, až když předchozí testovací fáze proběhly bez větších nedostatků. Testování provádí obvykle tým testerů na straně zákazníka dle scénářů připravených společnou prací zákazníka a dodavatele. Nalezené chyby jsou pak reportovány zpět dodavateli. Je důležité určit formu reportování chyb, jejich závažbosti, dobu na opravu chyby a limit pro počet nalezených chyb než jsou aplikovány sankce.

Podle uvedeného popisu feature souborů a vyjmenovaných druhů testů se čtenář jistě dovtípí, na které druhy testů se hodí používat testy psané pomocí Cucumber fraeworku. Ano jedná se o testy ve skupinách:

Funkční testování scénáře pro otestování požadované funkčnosti lze popsat do feature souborů a samotné testy lze také spouštět pomocí Cucumber frameworku.

Systémové testování podobný případ jako u funkčního testování. Složitější testovací scénáře lze popsat v feature souboru a jejich implementaci lze provést pomocí programovacího jazyka. O spouštění se následně postará Cucumber framework.

UAT - Akceptační testování tyto testy jsou obvykle realizovány pomocí týmu testerů, který na straně klienta nově dodanou aplikaci ručně proklikávají. Může se jednat o najatý tým testerů, nebo reálné uživatele systému, kteří podle předdefinovaných scénařů proklikávají aplikaci. Automatizace je tedy skoro nemožná a to z několika důvodů:

- Povětšinou se jedná o novou aplikaci, která se může ještě hodně měnit. Nemá tedy cenu připravovat automatizované testy.
- Testování probíhá na straně zákazníka, který nemá kapacitu pro psaní automatizovaných testů.

Tak Cucumber framework pomocí scénářů sepsaných do feature souborů může posloužit, alespoň jako předpis testovacích scénářů. A pokud bude potřeba, je možné později podle těchto feature souborů vytvořit automatizované testy.

Pro ostatní typy testů není předpis feature souborů vhodný. Testy jsou příliš technického rázu a netestuje se v nich přímo požadavaná funkcionalita.

Kapitola 3

Požadavky a analýza

Seznam požadavků s prioritami je uveden na interních wiki stránkách našeho hlavního projektu, v rámci kterého vzniká reportovací nástroj. Pro čtenáře neexistuje způsob, jak se přímo dostat k požadavkům zde uvedeným. Proto jsem seznam zkopíroval a dovolil jsem si jednotlivé požadavky více rozepsat a vytvořit z nich souvislý text, který čtenáři poskytne ucelenou představu o požadavcích na systém.

Požadavky v původním seznamu jsou rozdělené do logických celků a mají přidělenou prioritu. Priorita je číselná od 1 do n, kde menší číslo znamená vyšší prioritu. Momentálně veškeré požadavky na aplikaci mají prioritu v rozmezí 1 až 4, přičemž požadavky s prioritou 4 jsou mimo implementační rámec této práce a zmíním je jen pro zajímavost z pohledu budoucího rozvoje. O implementaci požadavků s prioritou 4 se pravděpodobně postará můj kolega v blízké budoucnosti. Stejně jako zbytek práce bude i implementace požadavků s prioritou 4 dostupná v githubovém repozitáři stengvac/cucumber-reports [4].

3.1 Rámcové zadání

Prvotní rámcové zadání znělo asi následovně: „ Pomocí webové aplikace prezentujte výsledky testů spouštěných pomocí Cucumber frameworku pro jazyk Java. Aplikace by měla poskytovat dostatečně detailní informace pro potřeby testerů a poskytovat potřebné statistiky, grafy a zobrazování trendů pro manažerské účely.“

Již z tohoto rámcového zadání bylo zřejmé, že bude potřeba využít pro ukládání dat databázové řešení.

Po prvních schůzkách a detailnějším prozkoumání potřeb jsem se dozvěděl, že potřebujeme ukládat pouze výsledky testů spouštěných pomocí nástroje Jenkins [3], který zajišťuje CI (Continuos integration) a CD (Continuous Delivery), nejen Javovských projektů. Sestavování projektů a následné spouštění testů bude probíhat na různých prostředích a bude proto nutné do výsledného řešení propagovat co nejdetailnější informace.

Tím vznikl požadavek na odesílání dat při průběhu testů, nebo po jejich dokončení na vzdálený server.

V pozdějších fázích projektu by měl systém odeslat mail zaiteresovaným osobám, jestliže se v sestavení objeví testy s chybami.

Zdálo se vhodné rozdělit aplikaci na několik částí podle jejich účelu. Jednotlivé části aplikace jsou popsané v kapitole návrh.

3.2 Prezentační sekce

Následující sekce obsahuje detailnější analýzu prezentační části aplikace. Aplikaci by mělo být možné konfigurovat skrze webové rozhraní.

3.2.1 Úvodní stránka

Úvodní stránka celé prezentační sekce by měla být jednoduchá stránka, kde si uživatel vybere jaké informace ho zajímají. Tím je myšleno, že je zde umístěný rozcestník do technické a statistické sekce.

3.2.2 Sekce pro technicky zaměřený personál

Tato sekce by měla obsahovat detailní informace o stavu jednotlivých sestavení. To znamená, že by uživatel měl mít možnost zobrazit informace o použitých proměnných při sestavování projektu, verzi projektu, použité git větvi, datu provedení sestavení, pořadové číslo sestavení a název projektu.

Aplikace by měla prezentovat výsledky ze spuštěných scénářů pomocí Cucumber frameworku v přehledné podobě s dostatečným detailem. Je žádoucí, aby s informacemi bylo možné exektivně pracovat při opravách testovacích scénářů.

3.2.2.1 Požadavky skrz celou sekci

Menu v celé sekci by mělo umožnit vyhledání sestavení. Vyhledávání má dva módy. Pokud uživatel zadá jen jméno projektu, pak aplikace zobrazí seznam všech sestavení pro tento projekt. Pokud uživatel zadá i sekvenční číslo sestavení, pak dojde k přesměrování na detail sestavení. Pokud nejsou nalezeny žádné výsledky, pak se uživateli zobrazí chybová hláška.

3.2.2.2 Úvodní stránka technické sekce

První stránka, kterou uživatel po příchodu do této sekce uvidí, by měla uživateli zobrazit seznam projektů, které v sobě obsahují testy napsané pomocí cucumber frameworku. U každého projektu by stránka měla obsahovat 10 nejnovějších sestavení a zobrazit ke každému sestavení minimálně informaci o tom, kdy proběhlo a jaké je jeho sekvenční číslo. Později přibyl požadavek, aby u každého sestavení bylo ještě uvedeno kolik procent testů prošlo. Tento

požadavek má několikero opodstatnění. Poskytuje osobě, pověřené správou testů, možnost se psychicky připravit na to, kolik je nutné opravit testů. Druhý souvisí s hranicí průchodnosti testů. Pokud je tato hodnota pod stanovenou hranicí je nutné jednat. Třetí důvod je, že pokud neprojde příliš mnoho testů, pak může být chyba někde jinde.

Po kliknutí na sestavení (jedné se tedy o odkaz) bude uživatel přesměrován na detail sestavení.

3.2.2.3 Detail sestavení

Tato stránka bude obsahovat detailní informace o jednotlivých sestaveních. URL stránky by mělo být neměnné pro snadnější opakovaný přístup.

Na stránce by se měly vyskytnout následující údaje o sestavení:

- seznam globálních proměnných, se kterými proběhlo sestavení projektu
- datum sestavení
- název projektu
- sekvenční číslo sestavení
- prostředí použité k průběhu sestavení
- git [18] větev, ze které pochází zdrojový kód

Zbytek stránky bude určený pro seznam sad s testy. Sada testů obsahuje testy spuštěné v rámci jednoho sestavení s různými proměnnými. Sady testů jsou potřebné, jelikož v rámci jednoho sestavení se mohou feature soubory spouštět opakovaně a to i se stejnou hodnotou glue (cesta k implementaci kroků). Bez rozdělení do testovacích sad by seznam s testovanými feature soubory obsahoval několik spuštění stejného feature souboru a nebylo by možné snado odlišit v čem se jednotlivé spuštění odlišují. Dobrým příkladem, proč se tak děje, je testování webových aplikací pomocí technologie Selenium [19]. Testy napsané pomocí Cucumber frameworku mohou testovat průchody webovými aplikacemi pomocí Selenia. Opakované spouštění je nutné pro otestování funkčnosti pro jednotlivé webové prohlížeče (Chrome, IE, Firefox..). Každé spuštění sady testů tak může mít jiné výsledky a může se spouštět i s trochu jiným nastavením (proměnná řídí, který prohlížeč se testuje) a je tedy nutné výsledky pro každou sadu oddělit.

Každý řádek sady testů je link na stránku s detailem sady (více informací v další kapitole). Link na detail sady by měl minimálně obsahovat jméno sady (nejspíše název proměnné, který odlišuje sady), procentuální úspěšnost průchodnosti testů v jednotlivých sadách.

3.2.2.4 Detail sady testů

Stránka, která obsahuje spuštěnou sadu testů v rámci jednoho sestavení. Sestavení může obsahovat jednu i více sad testů.

Stránka by měla obsahovat základní informace o sestavení uvedeném v předchozí sekci a také informace o sadě testů. Minimální informace poskytnuté ohledně sady jsou proměnné, se kterými byla sada testů spuštěna.

Ve spodní části obrazovky by měl být umístěn seznam spuštěných feature souborů v rámci této sady. Kliknutím na feature soubor se uživatel dostane na stránku s reportem testovaného feature souboru. Feature soubory by bylo vhodné seskupovat podle umístění feature souboru (cesta by pokud možno měla být unikátní v případě testů implementovaných v Javě reflektuje package ve složce test resources, kde se nachází feature soubor). U každého záznamu by mělo být uvedeno jméno feature souboru spolu s použitým glue (glue určuje kde se nachází implementace stepů) a počet průchozích scénářů ku celkovému počtu spuštěných scénářů. Počet spuštění scénářů je nutné počítat následovně. Spuštění sekce Scenario zvýší celkový počet spuštěných scénářů o 1 a spuštění sekce Scenario Outline navyšuje počet spuštěných scénářů o počet řádků v sekci Examples (každý řádek je spuštění scénáře s různými daty). Průchod sekcí Background nenavyšuje počet spuštěných scénářů.

Stránka by měla umožnit vyfiltrování pouze feature souborů, které obsahují scénáře s výsledkem rozdílným od úspěchu.

3.2.2.5 Report spuštění feature souboru

Tato stránka obsahuje report průběhu feature souboru. Stránka by měla uživateli poskytnout dostatek informací, aby byl v případě nastalé chyby ve scénáři schopný dohledat příčinu.

Vrchní část obrazovky by měla obsahovat základní informace o sestavení, jako je jméno projektu, sériové číslo sestavení a datum sestavení. Další informace ohledně sestavení jsou vítány. Zejména název prostředí, název git [18] větve. Vítané jsou i základní informace svázané s testovací sadou, ve které se reportovaný feature soubor nachází.

Zbytek obrazovky je určený pro prezentaci výsledků scénářů obsažených v feature souboru. Stránka by měla zobrazovat seznam názvů všech scénářů s možností vyfiltrovat jen scénáře, které obsahují chybu. Název scénáře se odvozuje následující metodou: Jako název scénáře se použije atribute name z feature souboru ze sekce Scenario respektive Scenario Outline. Pokud vnořená sekce Examples v sekci Scenario Outline deklaruje jako první sloupec s názvem DESCRIPTION, pak se jako název scénáře použije tato hodnota. Pro vícero průběhů scénáře je tak možné přiřadit různé názvy, které co možná nejlépe vystihují průběh testovacího scénáře. Jedná se o nestandardní požadavek, který vzešel z komplexnosti testů.

Sekce s reportem výsledků scénáře by měla obsahovat podrobné údaje o průběhu. Tady nejméně: Pokud je scénář otagován, pak je potřeba tagy zobrazit. V pozdějších fázích projektu bude vyžadováno vyhledávání podle tagů. Název scénáře (platí stejná pravidla jako jsou uvedena výše), všechny komentáře, které jsou obsaženy v těle scénáře a to u jakéhokoliv vnořeného elementu. Formátování komentářů by mělo zůstat zachováno.

Některé feature soubory mohou obsahovat Background sekci, kterou je také potřeba zobrazit u jednotlivých průchodů scénářů. Background sekce by měla obsahovat detailnější popis a další informace. Z Bacground sekce by mělo být zobrazeno jméno, popis a seznam kroků, které probíhají před spuštěním testovacího scénáře. V defaultním stavu Background sekce bude skrytá. Umístění Background sekce by mělo být v horní části reportu každého scénáře.

Metody anotované `cucumber.annotation.Before` nebo `cucumber.annotation.After` je nutné zahrnout do reportu. Pro přehlednost by metody anotované `cucumber.annotation.Before` měly být uvedeny před ostatními kroky scénáře i před kroky Background sekce a `cucumber.annotation.After` až na úplném konci. V defaultním nastavení by neměly být viditelné a jejich zviditelnění se bude řídit zaškrtnutím checkboxu v těle scénáře. Checkbox nebude viditelný, pokud scénář neobsahuje žádné `cucumber.annotation.Before` nebo `cucumber.annotation.After` metody.

Každý scénář musí obsahovat výčet kroků, které se spouští při automatickém testování. To zahrnuje i kroky obsažené v sekci Background, které se provádí před každým průběhem scénáře, avšak až po provedení metod anotovaných `cucumber.annotation.Before`. Čtenář si nejspíše položil otázku, jaký je rozdíl mezi metodami s anotací `cucumber.annotation.Before` a kroky v Background sekci. Hlavní rozdíl je, že Background sekce by měla obsahovat kroky pro vytvoření společného zázemí o kterých potřebuje autor a budoucí čtenáři feature souboru mít přehled. Naopak metody anotované `cucumber.annotation.Before` se ve feature souboru nevyskytují a čtenář feature souboru o nich nemusí vůbec vědět. Používají se spíše pro více technické úkony. Další rozdíl je, že do anotací lze přidat tagy a tím řídit, kdy se anotované metody mají spustit.

Kroky v Background sekci jsou stejně jako informace o Background ve výchozím nastavení schované. Zobrazení a skrytí se provede zaškrtnutím checkboxu a tím se u všech scénářů zobrazí informace o Background sekci včetně kroků. Checkbox pro zobrazení Background sekce je viditelný jen pokud feature soubor obsahuje Background sekci.

U všech kroků, jenž ve výsledku tvoří průchod scénáře (Before, Background, Scenario, After) je důležité zobrazit výsledek jejich průběhu. Možný výsledek je následující:

- passed - spuštění kroku proběhlo v pořádku
- failed - nastala chyba (zahrnuje i neimplementované a nenalezené kroky)

- skipped - všechny zbývající kroky ve scénáři, které následují po kroku, kde nastala chyba.

Výsledek kroků je barevně odlišený. U kroků je dále nutné zobrazit informace:

Pro Before a After metody zobrazit plnou cestu k metodě (v případě jazyka Java to znamená package.ClassName.methodName) a pokud možno i jména argumentů.

Pro Background a Scenario potažmo Scenario Outline kroky je nutné nejdříve zobrazit klíčové slovo (systém by měl podporovat klíčová slova i jiných jazyků než angličtina)

- Given
- When
- Then
- And
- But

následované samotným textem kroku. Pokud krok obsahoval záslepky, pak je nutné ze záslepek vytvořit odkaz, který po kliknutí nahradí záslepku reálnými daty a při dalším kliknutí opět hodnotu záslepky. Ve výchozím stavu jsou zobrazena reálná data a nikoliv záslepky.

Krok, který obsahuje chybu u sebe zobrazí stack vyjímky, kterou poskytuje Cucumber API a pokud je to možné, pak i odkaz k souboru nebo soubor samotný, který obsahuje zalogované informace ohledně průběhu sestavení.

Jednotlivé kroky mohou obsahovat vložené soubory nejčastěji vygenerované obrázky. Pokud nejaký krok (i Before a After metody) obsahuje obrázek, pak je u scénáře viditelné tlačítko, které zobrazí všechny vložené soubory vytvořené v průběhu scénáře.

Background a Scenario respektive Scenario Outline kroky mohou obsahovat tabulku s přidanými informacemi. Pokud krok takovou tabulku obsahuje, pak je vhodné ji u kroku zobrazit.

3.2.3 Statistická sekce

Tato sekce je určená zejména pro osoby, které zajímá statistické zpracování dat a trendy. Sekce by měla obsahovat statistiky pro jednotlivé stavení, vývoj hodnot v čase a trendy chování testů.

Tato sekce je z velké části mimo rozsah této diplomové práce a většina implementace započne až po dokončení sekce pro reportování zaměřenou na více technický personál. Důvod je jednoduchý. Nejdříve je potřeba data získat, aby bylo možné cokoliv zobrazit. Navíc aby grafy nebyly až moc červené, je potřeba mít nástroj pro údržbu scénářů. S větším množstvím dat je následně

již možné vybudovat smysluplnou statistickou část aplikace. Analýza tedy pro tuto část aplikace není ještě zcela hotová.

Ze statistické sekce je hotová jen jediná stránka a její popis je uveden v následující sekci.

3.2.3.1 Overview stránka

Tato stránka by měla obsahovat informace ohledně posledních sestavení všech projektů. Poskytuje tedy aktuální inforamce o vývoji Cucumber testů v projektech, které odesílají data na server.

Stránka by měla obsahovat dvě tabulky. První tabulku s informacemi o výsledcích spuštěných feature souborů. Konkrétně se jedná o informace o průchodnosti feature souborů, scénářů a stepů.

Druhá tabulka obsahuje informace čistě o počtu feature souborů, scénářů, kroků v projektu.

Kapitola 4

Dostupná řešení

Tato kapitola se zabývá rozborem dostupných řešení a snahou o pokrytí daných požadavků za jejich pomoci. Výsledné řešení může být složené z více dostupných řešení, pokud tak splní požadavky. Vzhledem k obsáhlosti požadavků je pravděpodobné, že ani kombinace dostupných řešení nepokryje všechny požadavky. V následujících sekcích uvedu pouze řešení, která lze použít pro jazyk Java a Jenkins.

4.1 Cucumber reporting

Cucumber reporting [20] je momentálně nejpoužívanějším řešením pro reportování výsledků, pro testy spouštěné pomocí Cucumber frameworku s CI nástrojem Jenkins [3]. Řešení používá pro vytvoření reportů JSON soubory, které jsou vygenerované pomocí JSONFormatter.java [21] a uložené na disk při běhu testovacích scénářů. JSONFormatter.java je nutné zaregistrovat do anotace CucumberOptions spolu s cílovou složkou a jménem vygenerovaného JSON souboru do atributu plugins.

Řešení se velmi často používá v kombinaci s pluginem pro Jenkins [22], který po dokončení sestavení prohledá složku s výsledky sestavení a použije JSON soubory, které vyhovují přednastaveným kritériím jako zdroj dat pro vytvoření HTML stránek. HTML stránky následně plugin uloží na disk se vším, co je potřeba k jejich zobrazení. Opět do složky, ve které proběhlo sestavení.

Pokud testy produkují nějaký druh vnořeného obsahu (nejčastěji obrázky vygenerované v průběhu testu - screen shot obrazovky atd.) je tento soubor obsažen přímo v HTML stránce. Načtení jedné stránky s větším počtem embedovaných souborů může být poměrně časově i pamětově náročné pro některé prohlížeče. Osobně jsem byl svědkem případu, kdy prohlížeč při načítání vygenerované stránky zamrzl.

Při použití Jenkins pluginu je odkaz na vygenerované stránky dostupný v Jenkins na stránce výsledku sestavení jako jedna z položek v menu.

Pro představu přikládám obrázek A.2 upraveného menu s Jenkins pluginem.

Úvodní stránka pro sadu reportů ze sestavení vypadá A.3. Stránka obsahuje tabulku s výsledky průchodu testů, scénářů a graf, který zobrazuje stejné informace. Ze stránky se lze prokliknout na detail jednotlivých feature souborů A.4. Stránka obsahuje detailnější informace o průběhu feature souboru a jednotlivých scénářů. Jak si čtenář může prohlédnout, uživatel musí rozklikávat jednotlivé sekce, aby se dostal k informacím a to není moc uživatelsky přívětivé.

Tento reportovací nástroj se momentálně používá i na našem projektu a s příchodem nové verze, kde se změnilo uživatelské rozhraní jsem začal od našeho testovacího týmu slýchávat, že doklikat se k příčině problému je často velmi zdlouhavé. Nová verze reportovacího nástroje totiž do HTML stránek zanesla rozklikávání vnořených elementů (momentálně není podporováno hromadné rozbalování/sbalování a tak se musí hezky po jednom) a to může být poměrně časově náročné a nepřehledné.

4.1.1 Shrnutí

Jelikož výsledky jsou ukládány formou JSON souborů na disku Jenkinse je složité následně provádět jiné operace než report aktuální sady testů. Hrozí ztráta dat, jelikož data jsou v různých složkách podle toho kde, se provedlo sestavení projektu a obvykle se informace starší x dnů mažou.

Nesplňuje z požadavků: Nejsou přítomné statistiky jednotlivých sestavení v čase a report feature souborů neposkytuje ani zdaleka potřebný detail informace.

4.2 Cucumber Test result plugin

Dalším uvažovaným řešením se stal Cucumber Test result plugin [23]. Tento plugin pro nástroj Jenkins přidává do Jenkins stránek základní náhled výsledků a průběhů testů. Stejně jako předchozí řešení využívá jako zdroj dat JSON soubory vygenerované pomocí JSONFormatter.java [21] a z nich jsou následně generuje HTML soubory. V popisu pluginu na github stránce [23] je uvedeno, že plugin vytváří grafy vývoje v čase, avšak v obrázcích použití pluginu jsem žádné screenshoty s takovou funkcionalitou nenalezl a ani pro krátkém zkoumání zdrojového kódu jsem nenalezl implementaci takové funkcionality.

Pro představu přikládám ukázky jak plugin vypadá. Na rozdíl od předchozího řešení jsou výsledky součástí stránek Jenkinsu. Stránka se přehledem A.5 obsahuje základní data o výsledých spuštění feature souborů a přítomných tag řetězcích. Stránka s edetailnějším reportem feature souboru A.6 obsahuje jen základní data z běhu.

4.2.1 Shrnutí

Tento projekt ani zdaleka nepokrývá požadavky na výsledné řešení. Stránky poskytují málo informací, jsou zabudované pevně do Jenkinsu. Statistická sekce chybí úplně. Navíc poslední push do projektu proběhl před cca 8mi měsíci a projekt od té doby působí, že není udržován.

4.3 Cucumber Performance Reports Plugin

Cucumber Performance Reports Plugin [24] je zameřený na výkon spouštěných Cucumber testů a to i v časové ose. Plugin si data tentokráte udržuje v XML souboru, do kterého následně podle potřeby zapisuje nové výsledky a čte výsledky pro prezentaci uživatelům. Stejně jako ostatní řešení získává data z JSON soubourů vygenerovaných při spuštění testů s existujícím formáterem.

Stejně jako v případě předchozího pluginu prezentuje plugin výsledky přímo jako součást Jenksins stránek. Plugin generuje celkem dva druhy stránek. První je vývoj doby trvání pro projekt přes jednotlivé sestavení A.11. Druhá obrazovka poskytuje údaje o nejhorších časech pro feature, scénář a krok A.12.

4.3.1 Shrnutí

Projekt neobsahuje stránky pro reportování výsledků feature souborů, ale poskytuje alespoň základ statistické sekce i když jen pro dobu běhu. Projekt by mohl ve výsledném řešení pokrývat s přimhouřením oka alespoň část požadavků na statisickou sekci. V github repozitáři je však napsáno, že projekt je momentálně neudržován.

4.4 Cucumber report DB

Cucumber report DB [25] jako jediné řešení používá pro uložení dat databázi. Kontrétně se jedná o MongoDB [26].

Pro získání dat používá vlastní formáter/reporter a odesílá data do MongoDB pomocí REST-API. Ze získaných dat pak webová část aplikace tvoří statistiky a resporty. Reporty však obsahují pouze se základní data.

Pro představu přikládám několik screen shotů aplikace. A.7 zobrazuje stránku určenou pro výsledky spuštění feature souboru. Obrazovka zobrazuje scénáře obsažené v feature souboru s výsledky jednotlivých kroků, které zahrnují status, délku běhu a vyprodukované přílohy. To je však vše, co aplikace o průběhu scénáře prozrazuje.

Druhý přiložený obrázek A.8 zobrazuje graf vývoje počtu průchozích/padlých scénářů v časové ose.

4.4.1 Shrnutí

Pro potřeby na projektu je však řešení nedostatečné. Poskytované reporty feature souborů obsahují opravdu pouze základní informace o průběhu a upřímně si nemyslím, že by podle nich bylo možné odhalit příčinu, proč test neprošel. Statistiky jsou pro naše účely bohužel také nedostatečné. Navíc poslední push do repozitáře byl před 6ti měsíci a projekt celkově působí opuštěným dojmem.

4.4.2 Závěr

Zvažoval jsem variantu projekt upravit, ale v podstatě by se jednalo o skoro úplné přepsání projektu a tak jsem tuto myšlenku opustil.

4.5 Bootstraped Multi Test Results Report

Bootstraped Multi Test Results Report [27] je projekt, který ze všech uvedených projektů působí po grafické stránce nejlépe. Jako zdroj dat pro vytváření reportů používá jako většina předchozích řešení JSON soubory vyprodukované pomocí JSONFormatter.java [21].

Z vyprodukovaných souborů projekt vytvoří HTML souboy. Projekt je nejčastěji použit ve formě Jenkins pluginu a ty uživateli zpřístupní skrze Jenkins rozhraní viz následující obrázek A.9.

Jako další přikládám pro představu screen shot ze stránky A.10, která zobrazuje výsledky z feature souboru. Projekt neposkytuje potřebný detail informace. V projektu navíc úplně chybí statistická sekce a vývoj jednotlivých hodnot v čase.

Další ukázky z aplikace lze najít [28].

4.5.1 Shrnutí

Reporty feature souborů neposkytují dostatečný detail. Řešení neobsahuje žádné statistiky.

4.6 Rozhodnutí

Na základě rozboru výše uvedených řešení jsem usoudil, že žádné řešení samo o sobě na pokrytí požadavků nestačí. Kombinace vícero řešení žádné velké zlepšení nepřinese. Jediná smysluplná kombinace je spojení Cucumber report DB a Cucumber Performance Reports Plugin. Tím by řešení kromě reportů feature souborů získalo ještě další statistiky. Problém je, že každé řešení používá jiný způsob pro ukládání dat. Aby řešení splňovalo požadavky z větší části, bylo by stejně nutné řešení v podstatě celé přepsat. Když posoudím pracnost s spojením dvou řešení (to zahrnuje i pochopit zdrojový kód obou projektů)

a přepsání do požadované podoby oproti implementaci řešení od začátku, tak jsem zvolil implementaci projektu na zelené louce.

Kapitola 5

Návrh

Jelikož systém vzniká takzvaně na zelené louce, bylo potřeba na úplném počátku vymyslet, jak systém bude navržen. Zdálo se vhodné systém rozdělit na několik spolupracujících částí. Každá část systému pak má za úkol řešit odlišnou část problematiky a v případě nutnosti spolupracovat s dalšími částmi systému. Systém je tak modulární a nahradit modul v případě potřeby by neměl být problém.

Systém jsme s pomocí kolegů nakonec rozdělili na tři části. Konečný návrh aplikace si můžete prohléhnout v přílohách A.13

5.1 Data collector

První a nejmenší část systému je koncipována jako velmi malá knihovna, která má za úkol napojit se na API Cucumber frameworku. Při průchodu testů knihovna z Cucumber API získává data o obsahu feature souborů a výsledcích průchodů testůvacích scénářů. Po shromáždění dat má modul za úkol už jen odeslat data ve formátu JSON pomocí HTTP POST požadavku na URL RESTového API, které data přijme a dále zpracuje. Myšlenka za tímto návrhem, je oddělit aplikaci, která zpracovává a zobrazuje data od jejich získávání. Je tak možné zaměňovat implementace serverové části a například podpořit i získávání dat z jiných jazyků než jen Java. K odeslání dat na REST API modul potřebuje celkem tři parametry, které získá z proměnných prostředí.

Data se odesílají po ukončení průběhu každého feature souboru. Data collector nemá možnost zjistit, jestli právě spuštěný feature soubor je poslední a je tedy na čase odeslat všechna data naráz nebo je ještě nutné počkat. To je způsobeno možností paralelního spouštění testů.

cucumber.job.name Jméno procesu, který se stará o sestavení projektu. Použije se pro seskupování výsledků zaslaných do REST API.

cucumber.job.sequential.number Pořadové číslo sestavení projektu. Každé spuštění sestavení má své unikátní číslo. Slouží k seskupování výsledků.

cucumber.rest.api.url URL, kam odeslat nashromážděná data. Je nutné zadat URL endpointu nejen doménu. Výsledné URL by mělo být ve tvaru <domain>/rest/collect.

Seznam podporovaných jazyků Cucumber frameworkem je k nahlédnutí ve zdrojích [6]. Díky oddělení modulu pro sběr dat a zbytku aplikace je možné podpořit získání dat i pro další programovací jazyky při zachování datového modelu.

5.2 Rest API

Druhá část aplikace, která vystavuje REST API má o něco složitější úkol.

Hlavním úkolem této části systému je zpracovat přijatá data od data collectoru pomocí vystaveného REST API. To se skládá z několika menších podúkolů. V prvé řadě je potřeba provést validace, zda přijatý JSON soubor a v něm obsažená data jsou validní. Systém by měl provést složitější validace, než jen validace typu JSR 303 [29]. Systém by měl tedy zkontrolovat, že přijatý JSON je validní feature soubor a obsahuje validní výsledky spuštění scénářů. Pokud data nejsou validní, pak by systém měl opět vrátit pomocí REST API JSON se seznamem chyb. Naopak, pokud data validní jsou, pak systém vyhledá kam zařadit nově příchozí data dle názvu projektu a sekvenčního čísla. Jelikož pro jedno sestavení může REST API obdržet mnoho požadavků, je potřeba implementovat mechanizmus pro seskupování příchozích požadavků pod jedno sestavení (pokud nelze přiřadit k existujícímu záznamu, pak vytvoří nový). Následně data persistuje s využitím DAO vrstvy.

Do budoucna se chystají další rozšíření pro REST API. Implementačně však nejsou v rozsahu této práce. Jsou plánované na další fázi projektu, která započala už v době psaní této práce. Plánovaná rozšíření jsou popsaná v následujících sekcích.

5.2.1 Poskytnutí ID

V dříve uvedených požadavcích je potřeba do systému zajišťujícímu CI a CD (v tomto případě Jenkins) vytvořit link přímo do reportu pro sestavení. V části systému pro zobrazení reportů je sestavení dostupné pod ID. REST API tak pro tyto účely poskytne ID sestavení aby bylo možné vytvořit odkaz.

5.2.2 Propagace konfigurace

Jednou ze slabin modulu data collector je, že data, která má shromažďovat z testovacího prostředí jsou v tomto modulu natvrdo zakódována. Vítané řešení je skrze REST API poskytnout data collectoru prefixy proměnných případně seznam názvů proměnných, které by měl data collector ukládat.

Konfigurace by se nastavovala pomocí webového rozhraní, takže by nebylo potřeba provádět žádné složité zásahy do systému nebo hromadné přejmenovávání proměnných na straně nástroje pro CI.

5.2.3 Odesílání emailů

Jeden z požadavků na systém je odesílání e-mailů zainteresovaným osobám. Tento požadavek bude nejspíše realizován pomoccí pluginu pro Jenkins. Princip by byl následující: na konci průběhu sestavení by plugin odeslal na RESTové rozhraní požadavek pro odesláním emailů pro projekt a sekvenční číslo. Logika umístěná v modulu by podle výsledků feature souborů rozhodla, zda je potřeba rozeslat emaily kvůli padlým testům.

Odesílání testů nelze řešit již při přijímání dat. Jedná se o stejný problém, jaký jsem uvedl u data collectoru a tím je, že systém nedokáže rozhodnout, zda již obdržel všechny výsledky pro sestavení. Je tedy nutný impuls zvenčí.

5.2.4 Poskytování dat přes REST

Momentálně se pro zobrazování nashromážděných dat uživatelům používá technologie na straně serveru. Do budoucna je však velmi pravděpodobné, že se tato technologie změní a nebo se vyskytne požadavek na poskytování dat třetím stranám.

Bude tedy nutné rozšířit REST API i o poskytování dat a to pro reporty i statistiky.

5.3 Prezentační část

Část systému, která koncovým uživatelům umožní prohlížet nashromážděná data pomocí webové aplikace.

Pro lepší přehlednost je sekce rozdělená na 3 části. Dělení nemusí být konečné. Rýsují se požadavky ohledně tendencí chování testů při releasu aplikací a možná tak nastane další dělení.

5.3.1 Sekce reporty

V této sekci jsou k nahlédnutí data nasbíraná z jednotlivých sestavení. Tato sekce je primárně určena pro osoby, které mají za úkol testy psát/spravovat/udržovat určitou kvalitu testů. Je to tedy spíše technicky zameřená část určená pro správu testů.

5.3.1.1 Statistická část

Naopak tato část je určená zejména pro osoby, které se zajímají o testy z pohledu spíše manažerského. Sekce nabídne statistiky pro jednotlivá sestavení nebo vývoj v čase a tedy umožní sledovat i tendence.

5.3.1.2 Admin část

Do této části aplikace by měli mít přístup jen administrátoři této aplikace. Bude zde možné konfigurovat adresáty emailů při nastalé chybě v sestavení, URL aplikací třetích stran, na které se aplikace bude odkazovat a možnost spravovat seznam projektů evidovaných systémem.

Kapitola 6

Realizace

Kapitola, která se zabývá realizací projektu. V kapitole je možné se dočíst více o použitých technologiích a průběhu implementace.

6.1 Technologie

Vzhledem k povaze projektu, pro který je projekt primárně určen, bylo ještě před začátkem projektu jasné, že využitý programovací jazyk bude Java a to ve verzi 8. Od toho se odvíjí i následný výběr dalších technologií použitých na projektu.

Jedním z cílů projekt je v rozumném stavu umístit projekt na github jako opensource řešení reportů a dále ho rozvíjet. Nejlépe použít technologie, které jsou open source a mají silnou komunitu. Vybrané technologie jsou důležité pro rozvoj projektu. Přeci jen technologie, které nikdo nezná nebo už se moc nevyužívají se nebudou potencionální kontributoři projektu učit jen kvůli nějakému open source github projektu.

6.1.1 Projekt

Projekt jako takový je veden jako maven [12] project. Maven poskytuje při vývoji SW v Javě spoustu užitečných ulehčení. Zejména se jedná o sestavení projektu, správu závislostí na knihovnách, dělení projektu do modulů. Jeho pomocí lze použít mnoho užitečných pluginů (například pro generování kódu).

Pro verzování kódu se využívá verzovací nástroj git [18]. Vývoj webové části aplikace probíhal na aplikačním serveru Jetty 8 [30].

6.1.2 Continuous Code Quality

Pro zajištění kvality kódu na projektu používáme SonarCube [31]. Jedná se o open source projekt, který provádí statickou analýzu zdrojového kódu ob-

vykle při pushi do centrálního repozitáře a podle definovaných pravidel pak tvoří reporty.

V reportech je možné pro jednotlivé projekty se proklikat až na úroveň tříd, které obsahují porušení definovaných pravidel. Porušení mají nastavenou závažnost a je nutné se pokud možno vyhnout závažným porušením a opravit je co nejrychleji pokud nastanou.

Tento přístup napomáhá udržovat požadovanou úroveň zdrojového kódu a předchází utváření špatných zvyků.

Pro představu uvedu několik pravidel pro kontrolu kódu. Další pravidla jsou k nalezení na [32].

- Nested blocks of code should not be left empty
- Unused "private"methods should be removed
- String literals should not be duplicated
- Expressions should not be too complex

Aplikovaných pravidel je však mnohem více. Finální číslo se pohybuje v řádech stovek až tisíců.

Další užitečnou vlastnostní nástroje je, že v něm lze určit quality gates. Tedy hranice, které musí zdrojový kód v projektu splňovat, aby projekt jako celek byl považován za dobře vedený. Pro představu uvedu několik požadovaných hranic ke splnění.

- Overall test coverage $> 70\%$
- Condition coverage $> 50\%$
- Duplicated code lines $= 0$ %
- Public Documented API $> 75\%$

6.1.3 Mapstrcuct

V aplikacích, které mají více vrstev a každá vrstva má svůj datový model, je nutné provádět konverze mezi datovými strukturami jednotlivých vrstev. Potřeba různých datových modelů souvisí s různými technologiemi, které každá vrstva používá a tedy i různými potřebami od datových modelů. Příkladem mohou být atributy, kterými aplikace řídí viditelnost prvků v uživatelském rozhraní. Takové atributy by nám v třídách, které reflektují databázový model užitečné nebyly.

Třídy, které zajištují tranformace mezi datovými strukturami mají v názvu obvykle postfix Converter nebo Mapper. Rozdíl mezi Converterem a Mapperem je v tom, že Converter zachovává při konverzi datových struktur všechny

informace. Mapper naopak mapuje některé zdrojové atributy na některé cílové. Dochází tak ke ztrátě informace, kterou nepotřebujeme. Dle mého názoru je dobrou praktikou psát Convertery i Mappery vždy pro jednu zdrojovou k ní cílovou datovou strukturu. To je sice velké množství tříd, ale jejich údržba a testování je mnohem snažší než udržování obrovských více účelových Converterů/Mapperů. Rozdělení do tříd je třeba brát s rozmyslem. Pokud se datová struktura vyskytuje pouze uvnitř jedné další struktury, pak je již na programátorovi, zda vytvořit Converter/Mapper i pro tuto třídu a využít nějakou z implementací návrhového vzoru Inversion of Control (inject instance objektu) a nechat si kontajnerem dopravit instanci třídy pro použití a nebo jen doplňit metodu s funkcionalitou.

Nejpíše si čtenář dokáže představit, že rozsáhlé projekty, které komunikují s okolím pomocí webových služeb nebo jiných informačních kanálů, potřebují transformovat velké množství datových struktur, které bývají poměrně složité. Často se tak stává, že se datové struktury mění a sem tam se na nějaký ten atribut zapomene. Pokud nemáme konverzní třídy dobře otestované, pak na chybu přijdeme až s odstupem času a to není dobré. Navíc psát konverzní metody bývá poměrně nezáživná činnost.

Z těchto důvodů jsem se rozhodl použít řešení, které generuje Converterey-/Mappery pomocí sady anotací na třídě a abstraktní konverzní metodě.

Mapstruct [33] je jedna z knihoven, která se snaží implementaci converterů/mapperů usnadnit. Rámcový princip: ve třídě nebo interface vytvořte abstraktní metodu, která bude sloužit k transformaci. Ideálně by metoda měla mít jako vstup zdrojovou datovou strukturu a jako výstup překonvertovanou datovou strukturu. Mapstruct zvládne i konverzi několika datových struktur do jedné, ale o tom více v dokumentaci mapstructu. Pomocí Java anotací, které určují pravidla transformace, knihovna Mapstruct při kompilaci zdrojových kódů vygeneruje tělo metody pro transformaci objektů. Pomocí anotací se definují pravidla pro konverzi. Mapstruct je poměrně chytrá knihovna a pokud se atributy ve vstupní i výstupní datové struktuře jmenují stejně, pak vytvoří mapování sám. Pokud chceme mapovat odlišná jména nebo některé atributy vynechat, je potřeba použít anotace. Mapstruct při generování kódu tak zkouší od top level atributů, zda je dokáže překonvertovat a pokud je potřeba, tak se zanořuje hlouběji. Vnořené datové struktury je možné konvertovat opět pomocí deklarace abstraktní metody. které mastruct vygeneruje tělo nebo poskytnutím konverzní třídy, kterou mapstruct použije.

Při použití mapstructu jsou hlavní výhody oproti ručnímu psaní takové, že všechny atributy na datových strukturách musí být namapované. Toto chování je vhodné nastavit nejlépe pro celý projekt. Defaultní hodnota je, že se při sestavení projektu vypíše chybová hláška, pokud se nepodaří atribut namapovat. Doporučil bych použít hodnotu, která ukončí průběh sestavení s chybou, dokud programátor mapování nedoplní. Další výhoda je, že se ušetří opravdu hodně času při psaní konverzních metod.

6.1.4 Log4j

Logování je nezbytnou součástí vývoje a následné údržby aplikací. Bez logování by v podstatě nebylo možné dohledat příčinu chyb při běhu aplikace na vzdáleném serveru.

Log4j [34] je velmi široce používaná knihovna, která umožnuje logovat chod aplikace bez znatelného dopadu na výkon při běhu aplikace. Logování je možné navíc rozdělit na různé urovně a označit tak důležitost logu.

6.1.5 FasterXML Jackson

V aplikaci se vyskytla potřeba mapování mezi POJO a JSON formátem. Bylo tedy nutné využít knihovnu pro mapování. Pro tyto účely lze použít knihovnu FasterXML Jackson [35]. Díky této knihovně lze transformovat objekty velmi snadno, jen za použití třídy ObjectMapper [36] nebo její registrací v frameworku, který se o mapování postará interně. Knihovna poskytuje možnost POJO anotovat a tím měnit názvy atributů v JSON formátu, filtrovat JSON atributy, určovat jejich pořadí atd. Knihovna také nabízí slušnou podporu pro práci s Time package [37]. Lze tak definovat neměnné REST API v podstatě bez ohledu na to, jak se jmenují atributy v POJO a jaké datové struktury používáme.

6.1.6 Spring

Při výběru frameworku pro vývoj po dlouhém boji mezi J2EE 7 [38] s CDI 1.1 [39] vs Spring 4.3.7+ [40] vyhrál Spring. Volba byla velmi podstatná, jelikož se v podstatě jedná o konkurenční technologie. Pokud se vybere špatně, tak přechod na druhou stranu je velmi obtížný.

Hlavní důvody jsou:

1. J2EE se přeci jen už nějaký ten rok aktivně nevyvíjí, zatímco Spring je aktivně ve vývoji.

2. Spring nabízí pro MongoDB velmi dobrou podporu ve formě knihovny SpringData [41].

3. Spring je v komunitě Java Developerů populární a známý. Nebude tedy problém s neznalostí technologie.

4. Spring je open source projekt se širokou komunitou kontributorů. Nehrozí tedy, že by projekt byl opuštěný.

Využití frameworku Spring přináší celou řadu výhod a urychlení vývoje. Jednotlivé moduly Springu, které jsou v projektu použité jsou sepsány v následujících sekcích spolu s účelem jejich použití.

6.1.6.1 Spring core

Hlavní část Springu, která přináší do aplikace zejména DI. DI alias dependency injection je jedna z implementací IoC (Inversion of Control), které využívají Hollywood pricip „Don't call us, we'll call you" [42]. Tohoto principu následně využívá DI a programátor přenechává starost o vznik instance a propagace instance v programu na frameworku. Díky tomu není nutné tvořit singletony pomocí statických proměnných a celkově se tak zlepšuje čitelnost, udržovatelnost a testovatelnost kódu.

V projektu je použita verze Spring core 4.3.7+, která umožňuje řídit DI pomocí anotací v kódu, místo dříve používaného způsobu přes XML soubory. Tento přístup je programátorsky přívětivější a hlavně přehlednější.

6.1.6.2 Spring webmvc

Modul Springu, který usnadňuje tvorbu webových aplikací. V aplikaci je tento modul využitý hlavně pro vytvoření RESTového API pomocí Springových RestControllerů a handlerů vyjímek pomocí Advice tříd. Další využití je v části aplikace pro zobrazování výsledků, kde je nutné zaregistrovat resolver expression language pro použitý server side templetetovací framework JSF [43] viz 6.1.7.

6.1.6.3 Spring security

Modul Springu určený pro zabezpečení aplikací. V aplikaci je momentálně zabezpečená pouze sekce s konfigurací aplikace pomocí HTTP Basic security.

6.1.7 Java Server Faces

V dnešní době existuje velká řada technologií pro tvorbu dynamických webových stránek pomocí šablonovacích enginů a nebylo tedy jednoduché si zvolit. Na výběr je od starších, avšak prověřených server sidových technologií, které používají expresion language a Javu pro vyrenderování stránky, až po poměrně nové client sidové javascriptové frameworky jako je Angular [44] a React [45].

Finální výběr probíhal mezi třemi technologiemi. Jmenovitě se jednalo o:

1. JSF - Java server faces. Server sidový framework pro renderování XHTML stránek pomocí Javy, template souborů s expression language. JSF nejsou závislé na J2EE ani Springu, avšak mají přeci jen o něco lepší podporu na straně J2EE a to hlavně díky knihovně Omnifaces [46].

2. Wicket [47] - Server side framework, který používá template soubory pouze pro namapování HTML komponent napsaných v Javě na výslednou HTML stránku. Veškorá logika svázaná s chováním aplikace je napsaná v Javě. To je jeden z hlavních rozdílů oproti JSF. JSF oproti Wicketu umožňují konfigurovat komponenty v XHTML souborech.

3. React - Java scriptový client side framework od Facebooku.

Angular 1 ani 2 v seznamu neuvádím, jelikož jsem absolvoval cca 2,5 hodiny dlouhé sezení s týmem front end vývojářů, kteří pracují na front endu pro AirBank [48]. Prezentovali zde své zkušenosti během posledních několika let vývoje v Angularu 1, přechod na Angular 2 a s tím spojené slasti a strasti. Spíše se jednalo o strasti. Pro představu uvedu několik důvodů proč jsem Angular nezařadil do dalšího výběru.

6.1.7.1 Nevýhody Angular 1

- Google již aktivně Angular 1 nevyvíjí, jen zajišťuje bug fixing.
- Framework využívá two way data binding. Jedná se o to, že změna dat od uživatele se ihned propaguje do modelu a změny v modelu se propagují uživateli a to okamžitě po změně. V podstatě se data mění ze dvou zdrojů. To může způsobovat neočekávané chování a špatně se odhalují chyby tímto chováním způsobené.

6.1.7.2 Nevýhody Angular 2

- Velmi odlišný od Angular 1. Až natolik odlišný, že není kompatibilní s Angular 1 a aplikace napsané v Angular 1 je nutné z velké části upravit než je mžné použít Angular 2.
- V době sezení s Airbank frontend týmem měl několik release kandidátů a každý z nich obsahoval velké změny nekompatabilní s ostatními release kandidáty.

Ať neuvádím jen nevýhody. Angular 2 umožnuje použití TypeScript [49] pro psaní aplikací. To je velká výhoda oproti Angular 1.

A další verze Angularu vznikají dle mého názoru jak houby po dešti. Myslím, že není na škodu ještě nějakou dobu počkat, než se vývoj aspoň trochu uklidní a bude stabilní verze bez velkých změn v každém release.

6.1.7.3 React

Podle popisu AirBank frontend týmu je react jasná volba při požadavcích na Javascriptový client side framework (ještě aby ne, když se v něm píše Facebook). Nakonec jsem se však k této technologii nepřiklonil kvůli časovým důvodům. Naučit se pracovat s novým frameworkem vyžaduje obvykle poměrně hodně času a ten jsem bohužel neměl. I když se na to chystám v brzké době.

A tak jsem se uchýlil k výběru z technologií, se kterými jsem již v minulosti pracoval.

6.1.7.4 Wicket vs JSF

Konkrétně se jedná o frameworky Wicket a JSF. S oběma frameworky jsem pracoval přibližně stejně dlouhodu dobu a mé znalosti jsou pro oba frameworky poměrně rozsáhlé.

Oba frameworky jsou server sidové a napsané v Javě. Hlavní rozdíl je v přístupu k problematice. Wicket je v podstatě čistá Java a naopak JSF přenáší spoustu konfigurace HTML komponent do šablon. Další rozdíl je v práci s modely, které plní stránky a kooperace mezi komponentami. Zde jasně vítězí JSF. Ve Wicketu je potřeba složitě propagovat reference na jiné komponenty a používat dědičnost a hlavně anonymní třídy ve velkém (na tom je ale Wicket založený). Celkově mi přijde při použití JSF zdrojový kód čistější než v případě Wicketu. Na druhou stranu šablony pro renderování webových stránek napsané v JSF mohou být poměrně nepřehledné. To už je však o přístupu.

Zvolil jsem tedy JSF a přibral k němu knihovnu Primefaces [50].

6.1.7.5 Primefaces

Knihovna Primefaces [50] je bohatá knihovna pro framework JSF. Obsahuje mnoho předpřipravených komponent, které je možné ihned použít. Obsahuje i podpůrnou funkcionalitu pro vývoj aplikací - vyhledávání komponent, obnovení stránky.

Předpřipravené komponenty jsou opravdu velká výhoda, pokud pracujete v JSF a opravdu se hodí mít knihovnu, která poskytuje dobré komponenty.

6.1.8 MongoDB

Při rozhodování, jakým způsobem a kam budou data persistována, bylo nutné brát v potaz několik faktů.

6.1.8.1 Je třeba použít DB?

To byla první otázka, kterou jsem si kladl při úvahách o tom, jak persistovat data. Většina existujících řešení pro reportování používá jako zdroj informací soubor formátu JSON. Ten je obvykle uložený na disku ve složce s výsledky sestavení. Kromě zobrazování reportů pro jednotlivá sestavení však má systém umožnit zobrazení statistik s vývojem v časové ose. Testy jsou obvykle spouštěné na vzdáleném serveru pomocí nástroje pro CI a CD, který obvykle adresáře starších sestavení po uplynutí stanoveného časového intervalu smaže kvůli zbytečně zabranému místu. Některé CI a CD nástroje umožňují mazání dat vypnout, avšak při přesunu na jiný stroj se data mohou ztratit.

Z důvodu zobrazení statistik a mazání informací starších sestavení jsem se uchýlil k použití databáze. Při běhu aplikace nepředpokládám silnou zátěž na DB a tam nebylo potřeba použít žádné silné komerční databázové řešení.

6.1.8.2 Použitá DB

Při návrhu aplikace se nepočítalo s velkou zátěží na DB. Počet sestavení, která v sobě obsahují testy napsané pomocí Cucumber frameworku jsou i na velkých projektech maximálně v řádech stovek za den.

Odhad zobrazení reportů je podobný jako počet spuštění sestavení projektu. Reporty jsou primárně využity, když sestavení obsahují testy s chybami. Tedy opět maximálně v řádech stovek za den.

Odhad vytíženosti statistické sekce je v podstatě stejný jako u předchozích sekcí.

Podle předchozích úvah tedy není nutné použít extrémně výkonnou DB.

Nakonec jsem se rozhodl použít MongoDB [26] s podporou frameworku SpringData. MongoDB je dokumentová databáze (NoSQL) s maximální velikostí dokumentu 16MB ve formátu JSON.

Výhoda: existující řešení využívají povětšinou také formát JSON a je možné, že se využijí i vygenerované JSON soubory aspoň pro statistické účely.

6.1.9 Podpůrné knihovny

Často se v projektech vyskytují různé podpůrné knihovny, které se postupem času staly v podstatě již nepostradatelnou součástí všech projektů. Tak je tomu i u tohoto projektu.

6.1.9.1 Guava

Guava [51] je knihovna od Google, která obsahuje opravdu mnoho užitečných tříd pro vývoj aplikací. Pro představu mohu uvést Preconditions - funkcionalita s validací proměnných, Itetables - práce s kolekcemi, Streams podpůrné funkce pro práci s Java 8 Stream APi a další.

6.1.9.2 JSR 303 - Bean Validation

Uživatelské vstupy je potřeba validovat a psát validace ručně není to pravé. Mnohem lepší alternativou je k tomu použít knihovní validační anotace specifikované ve specifikaci JSR 303 [29]. Na projektu je použita jejich implementace ve formě hibernate-validator [52] knihovny.

6.1.10 Testování

6.1.10.1 jUnit

Pro unit testování v aplikaci je snad již standardem využít knihovnu jUnit [13], která pomocí sady anotací pro běh testů a metod pro ověření výsledků poskytuje slušné zázemí pro testování aplikací na úrovni tříd.

6.1.10.2 Mockito

Při jednotkovém testování je potřeba veškerou komunikaci, vně třídy a hlavně reakce, nahradit falešnými odpovědmi. Pro tyto účely existuje hned několik knihoven, ale již na několika projektech pracuji s knihovnou Mockito [53] a tak jsem ji použil i zde. Přestože Spring framework nabízí modul pro testování i tak stále preferuji Mockito.

6.2 Postup práce

6.2.1 Agilní vývoj

Vyvinout aplikaci čistě na základě požadavků, které jsou známy ještě před počátkem vývoje, je dnes prakticky nemožné. Vždy se na něco zapomene, je potřeba přidat a nebo to vlastně není potřeba. Proto se dnes často uchyluje k využití agilních metodik pro vývoj SW. Tak tomu je i v případě tohoto projektu. Kontkrétně byl použit Scrum [54] v ořezané podobě a to kvůli velikosti týmu.

6.2.2 Scrum

Jedné se asi o nejpoužívanější agilní techniku pro vývoj SW. Oproti normálnímu použití Scrumu se pro vývoj aplikace použila pouze jeho část.

6.2.2.1 Product backlog

Jedná se o seznam úkolů seřazených podle priority a rozdělených do kategorií. Seznam je obvykle umístěný na dobře viditelném místě a tím tým, který se stará o implementaci, může snad pozotovat pokroky při implementaci. Také je tento seznam velmi populární u manažerů. Rozdělení úkolů do kategorií může vypadat například následně:

- Funkcionalita
- Technické úkoly
- Chyby
- ...

Tato struktura není směrodatná a může se tady lišit projekt od projektu. Obecně vzato v tomto seznamu musí být obsaženo vše, co je potřeba udělat, aby výsledný produkt byl kompletní. Vlastník produktu následně určuje pořadí úkolů. Priorita i seznam úkolů se mohou během vývoje měnit. Často se tak děje po demu naimplementované části aplikace.

Od samotného počátku implementace v produktovém backlogu bylo mnoho požadavků na funkcionalitu s prioritou 1 až 4, kde úkol s menším číslem má

větší prioritu. Požadavky jsou uvedeny zde požadavky a analýza. Co se týče technických úkolů, několik jich bylo známo od počátku, avšak povětšinou vznikaly až v průběhu implementace.

I v současné době se product backlog rozrůstá o další požadovanou funkcionalitu.

6.2.2.2 Sprint

Iterace pro implementaci části funkcionality. Začíná plánováním, kde se vytvoří z product backlogu spring backlog (vybere se část úkolů k implementaci) a končí retrospektivou pro zhodnocení sprintu. Při práci na projektu měl sprint trvání 14 dní, přičemž jsem měl na práci na projektu vyhrazeno max 5 MD.

6.2.2.3 Spring backlog

Seznam úkolů, které je třeba implementovat během sprintu. Seznam úkolů pro sprint pochází z úkolů uvedených v product backlogu. Přižemž se berou úkoly s nejvyšší prioritou. Sprint backlog můžeme doplnit o úkoly technického rázu pokud jsou potřebné. Při plánování jsou moc velké úkoly rozděľené do menších celků.

6.2.2.4 Product Owner

Osoba zodpovědná za stav product backlogu. V průběhu implementace zastával tuto funkci vedoucí SIT týmu.

6.2.2.5 Scrum Master

Osoba zodpovědná za správné použití Scrumu pro maximalizaci výkonu. V průběhu implementace tuto roli zastával scrum master týmu, ve kterém pracuji na ostatních projektech.

6.2.3 Planning

Na počátku každého sprintu se naplánovalo, co se za daný sprint udělá podle priorit úkolů v backlogu.

6.2.4 Sprint review

Na konci sprintu obvykle bylo review, kde jsem svému Scrum Masterovi sdělil, co jsem ve sprintu stihl a nestihl. Demo obvykle nenásledovalo. Bylo domluvené po delších časových úsecích.

6.2.5 Prezentování výsledků

Prezentace implementovaných výsledků probíhala ve formě prezentace na demu. Demo bylo určené pro tým testerů a product ownera. Testeři v tomto kontextu zastupují uživatele reportovací části systému, přičemž product owner se zajímal spíše o statistickou část.

6.2.5.1 Vynechané části a role

Při implementaci nebylo nutné využít Daily Scrum, kde členové týmu sdělují zbytku týmu to, co za poslední den udělali, co plánují a případně s čím mají problémy. Jelikož jsem v drtivé většině času tvořil implementační tým jen já, tak tato praktika postrádala smysl.

Další vynechaná část je retrospektiva. Zhodnocení, které mělo za účel shrnout co se povedlo/nepovedlo a jak proces zlepšit jsem provedl v rámci svého svědomí.

6.2.6 Průběh implementace

V následující kapitole se pokusím nastínit, jak postupně probíhal vývoj a jaké okolnosti vedly k různým rozhodnutím. U některých částí implementace zmíním práci mého kolegy ze zaměstnání, který se na počátku také podílel na implementaci. Jeho kód z větší části již není součástí aplikace, až na několik tříd v DAO vrstvě a view modulu.

Pro čtenářovu představu se pokusím text doplňit o komentované ukázky zdrojového kódu.

Předpokládejme, že již proběhla rámcová analýza a návrh aplikace. Bylo tedy možné začít se samotnou implementací.

Nejdříve bylo potřeba nastudovat reportovací API Cucumber frameworku. API poskytuje údaje pomocí dvou rozhraní. První je Reporter rozhraní [55], který poskytuje výsledky spuštění jednotlivých kroků, before a after metod a vygenerovaného vnořeného obsahu. Druhé rozhraní je Formatter [56], který poskytuje informace o prvcích feature souboru jako takového. Z dokumentace a pomocí opakovaného spouštění testovacích feature souborů v debug módu jsem získal poměrně dobrou představu o životním cyklu spouštěných testů, poskytovaných datech a chování v krajních situacích. Po podrobnějším seznámení s datovou strukturou poskytnutou API bylo možné vytvořit potřebné datové struktury pro jednotlivé vrstvy aplikace. Konkrétně se jedná o transportní model určený pro odeslání dat na RESTové API v JSON formátu, který se v budoucnu stane podkladem pro popis RESTového API i pro jiné aplikace či implementace data collectoru pro další programovací jazyky. Databázové entity a později i model objekty ve view modulu určené pro prezentaci dat. V této fázi však prozatím stačilo vytvořit transportní model a databázové entity.

Po vytvoření modelů mohlo započít samotné napojení na reportovací API Cucumber frameworku. Tedy implementace modelu a extrakce dat z Cucumber rozhraní. Extrakce dat nebyla úplně triviální záležitost, jelikož API například nerozlišuje výsledky kroků v background sekci a scénářích. Bylo tak nutné doimplementovat potřebnou logiku. Dále následoval návrh potřebného REST API rozhraní pro shromaždování dat a implementace. V průběhu implementace odesílání dat na REST API vyvstávaly různé otázky. Jedna z nich byla, zda se dá poznat, že doběhl poslední test a odeslat tak data najednou, nebo je nutné vymyslet jiný princip. Nakonec se ukázalo, že testy lze spustit paralelně a nedá se dost dobře určit, zda se jedná o poslední průchod testem. Data se tak odesílají po dokončení testování feature souboru. Tuto informaci poskytuje API Cucumber frameworku. Pro indikaci dokončení testování je tak nutné implementovat plugin pro CI a CD nástroj (Jenkins), který opět přes REST oznámí konec sestavení. Ale tato funkcionalita bude potřeba až později.

Ve finálním řešení se na RESTové API odesílají data po ukončení běhu každého feature souboru spolu s informacemi o probíhajícím sestavení. Odesílat data tímto způsobem je nutné, jelikož testy lze spouštět paralelně a nelze tak určit jejich pořadí. Data nelze odeslat ani hromadně po ukončení spuštění posledního testu, jelikož jUnit runner nemá informaci o tom, který test je poslední. Tuto informaci by bylo nutné získat z externího zdroje a to by dělalo data collector závislý na externím nástroji. Logika na straně serveru se tak musí postarat o umístění přijatých dat do správného sestavení, nebo vytvoří nové za pomoci obsažených dat.

Informace o konci sestavení bude bohužel potřeba pro pokrytí požadadavku na odesílání emalů osobám, které mají na starost testy v sestavení - email odeslat jen 1x, když už jsou dostupná všechna data. Ale pro tuto feature vznikne pro CI nástroj plugin, jelikož odesílání emailů nemusí využít všechny projekty.

Po dokončení modulu data collector následovala práce na REST API a DAO vrstvě. V prvotní verzi REST API byl modul, který jen přjímal data a pomocí konvertorů je transformoval na DAO entity.

Postupem času REST API modul pojmul další funkcionalitu. Konkrétně se jedná o validaci příchozích TO a odeslání chyb zpět ve formátu JSON.

Další na řadě bylo implementovat persistenci nashromážděných dat v DAO vrstvě. Bylo tedy nutné nastudovat jak pracovat s MongoDB skrze SpringData a vytvořit DB modely.

Samozřejmě všechny datové struktury bylo nutné konvertovat. Než jsem se dozvěděl o mapstructu, tak již valná část konverterů byla napsaná a tak jsem provedl refaktoring a convertery/mappery přepsal pomocí mapstructu. Úbytek kódu byl znatelný. Přeci jen pokud se atributy zdrojové i cílové datové struktury shodují, je nutné zadeklarovat pouze abstraktní metodu a dodat prostředky pro konverzi vnořených netriviálních datových struktur viz příklad.

Datová struktura ke konverzi.

Code 6.1: Zdrojová dat. struktura pro konverzi

```
package cz.airbank.cucumber.reports.transport.model;

import java.util.ArrayList;
import java.util.List;

public class HookDefinitionSrc implements Serializable {

        private String location;
        private List<ArgumentSrc> arguments;

        //getters and setters
}
```

Cílová datová struktura.

Code 6.2: Cílová dat. struktura pro konverzi

```
package cz.airbank.cucumber.reports.dao.entity;

import java.util.ArrayList;
import java.util.List;


public class HookDefinitionTar {

        private String location;
        private List<ArgumentTar> arguments;

        //getters and setters
}
```

Datové struktury `ArgumentSrc`, `ArgumentTar` záměrně neuvádím, protože obsahují jen Stringové atributy, které se jmenují stejně v obou strukturách. Mapstruct tedy dokáže vygenerovat obsah konverzní metody jen s touto znalostí.

Všechny vygenerované konverzní funkce obsahují kontrolu `null` a vrací pro ni opět `null`.

Code 6.3: Interface pro budoucí converter - Argument

```
@Mapper(
        componentModel = "spring",
        unmappedTargetPolicy = ReportingPolicy.ERROR
)
public interface ArgumentSrc2ArgumenTarConverter {
        ArgumentTar convert(ArgumentSrc source);
}
```

Povšimněte si, že annotace `@Mapper` obsahuje atribut `uses`. Ten obsahuje seznam converterů, které má anotovaný converter použít při generování konverzní metody. V tomto případě tedy využije při generování converter pro datovou strukturu reprezentující argument. Mapstruct také dokáže vygenerovat konverzní metody pro seznamy, pokud dokáže provést konverzi jednoho prvku.

Příklady dále obsahují konfiguraci pro použití poskytnutých convertorů. V tomto případě je hodnota `componentModel` rovna "spring"a convertery tak budou do výsledné implementace injektovány pomocí anotací Spring frameworku. Atribut `unmappedTargetPolicy` slouží pro konfiguraci, jak se má mapstruct zachovat pokud se mu nepodaří namapovat atribut. S touto konfigurací kompilace skončí neúspěchem. Oba parametry lze nakonfigurovat pro celý projekt v globálním měřítku.

Code 6.4: Interface pro budoucí converter - HookDefinition

```
@Mapper(uses = ArgumentSrc2ArgumenTartConverter.class,
        componentModel = "spring",
        unmappedTargetPolicy = ReportingPolicy.ERROR
)
public
interface HookDefinitionSrc2HookDefinitionTarConverter {
   HookDefinitionTar convert(HookDefinitionSrc source);
}
```

Předchozí ukázky při sestavení projektu vygenerují v defaultním nastavení třídy s postfixem Impl s implementací abstraktních metod.

6.2.7 Prezentační modul

Před samotnou implementací prezentační vrstvy bylo nutné navrhnout strukturu stránek a vytvořit několik wireframů pomocí nástroje Balsamiq [57]. Pomocí wireframů jsme s kolegou prezentovali testerskému týmu, jak by mohla budoucí aplikace vypadat. Náměty a připomínky jsme zapracovali a začali jsme pracovat na prezentační vrstvě.

Kolega pracoval na statistické části, která zobrazuje počet testů v sestavení a jejich výsledky ve formě dvou tabulek. Do dnešního dne je to jediná stránka ve statistické sekci. Ja jsem si vzal na starost sekci technicky zaměřenou a implementoval reportování nashromážděných dat. Později také část sekce pro konfiguraci aplikace. Detailnější analýza a návrh této sekce je popsaná v kapitolách požadavky a analýza a návrh. Implementace proběhla bez větších problémů za pomoci Springu, JSF, Primefaces a Bootstrapu [58].

6.3 Ukázka vybraných obrazovek aplikace

Pro představu o výsledku jsem ze současného stavu aplikace vytvořil několik snímků obrazovek. Data zobrazená v aplikaci jsou vygenerovaná spuštěním feature souboru přiloženého na CD ve složce `src/thesis/features`.

První snímek zachycuje úvodní obrazovku celé aplikace A.14, kterou uživatel uvidí jako první obrazovku při přechodu na URL s aplikací. Stránka umožňuje přechod do statistické sekce nebo reportovací sekce podle potřeb uživatele. Druhý snímek je první obrazovka reportovací sekce. Snímek A.15 zobrazuje obrazovku, která zobrazuje posledních 10 sestavení pro každý sestavovaný projekt, který obsahuje Cucumber testy. Třetí snímek A.16 reprezentuje obrazovku s detailem testovací sady, která je součástí sestavení na druhého snímku. Uživatel se na tuto obrazovku dostane kliknutím na sestavení z druhého snímku. Klikem na jméno feature souboru ve třetím snímku se uživatel dostane na obrazovku, která slouží pro reportování feature souborů. Reportovací stránka je zastoupena dvěmi obrázky. První zobrazuje vrchní část obrazovky s přehledem testovacích scénářů A.17. Druhý snímek A.18 zobrazuje detailněji neprůchozí scénář s chybou.

Statistická a konfigurační sekce nejsou v ukázkách zastoupeny.

Kapitola 7

Instalace aplikace

Vítejte v instalačním průvodci aplikace. Momentálně ještě neexistuje stránka, která by poskytla ke stažení přímo WAR soubory. Je tedy nutné vyklonovat projekt z repozitáře stengvac/cucumber-reports, [4] nebo použít kód na přiloženém CD. Doporučil bych použít githut repozitář.

7.1 Databáze

Pro běh aplikace je potřebná MongoDB ve verzi 3.2 nebo vyšší.

V aplikaci je možné nastavit připojení k databázi skrze soubor `application.properties`, který se nachází v dao modulu ve složce resources.

V souboru je možné konfigurovat několik proměnných souvisejících s nastavením databáze. Nastavení jsou uvedena s defaultními hodnotami, které se použijí pokud nic nezměníte.

Code 7.1: Konfigurace MongoDB

```
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.user=null
spring.data.mongodb.password=null
spring.data.mongodb.database=cucumber_reports
```

7.2 Požadavky na aplikační server

Applikační server pro běh aplikace musí dokázat pracovat s JRE 1.8. Je možné použít například Tomcat Server 8 [59].

7.3 Instalační proces

Zdrojové kódy jsou ke stažení v repozitáři stengvac/cucumber-reports [4].

Na lokálním počítači s pomocí příkazu `mvn clean install` spuštěném v kořenovém adresáři projektu připravte projekt k použití.

Pokud se předchozí krok povedl, pak adresáře view a rest-api v jejich podadresářích target budou obsahovat soubor *.war. Tyto dva soubory je nutné nasadit na dříve zmíněný aplikační server.

V brzké době se pokusím umístit na internet oba war soubory ke stažení, aby uživatel WAR soubory jen stáhl a nasadil na aplikační server.

7.4 Použití

Otevřete projekt, ze kterého chcete odesílat na server data z testů.

Pokud se jedná o maven projekt, přidejte závislost do pomu na data collector modulu. Modul zatím není dostupný v centrálním maven repozitáři. Novinky v této oblasti budou zaznamenány v github repozitáři. Přidání závislostí na aktuální verzi projektu může vypadat následovně.

Code 7.2: Závislosti na data-collectoru

```
<properties>
  <!-- Current version-->
        <cucumber.reports.version>
         1.3.2
        </cucumber.reports.version>
</properties>

<dependency>
        <groupId>cz.airbank.cucumber.reports</groupId>
        <artifactId>data-collector</artifactId>
        <scope>test</test>
        <version>${cucumber.reports.version}</version>
</dependency>
```

Javovské třídy, které spouští feature soubory jsou anotované `@RunWith(Cucumber.class)` a je nutné nahradit je za `@RunWith(UnoCucumberRunner.class)`. Třída `UnoCucumberRunner` rozšiřuje původní použitý runner a přidává další plugin, který sbírá potřebná data. Runner pouze přidává plugin a je tedy možné použít i další pluginy pro vytvoření odlišných výstupů pro jiná řešení. Upravený runner odesílá data na REST API pouze pokud jsou nastavené tři proměnné prostředí uvedené v seznamu proměnných prostředí. Pokud nastavené nejsou, pak se `UnoCucumberRunner.class` chová přesně jako původní runner. Je tak možné s upraveným runnerem pracovat i lokálně.

cucumber.job.name Jméno procesu, který se stará o sestavení projektu. Použije se pro seskupování výsledků zaslaných do REST API.

cucumber.job.sequential.number Pořadové číslo sestavení projektu. Každé spuštění sestavení má své unikátní číslo. Slouží k seskupování výsledků.

cucumber.rest.api.url URL, kam odeslat nashromážděná data. Je nutné zadat URL endpointu nejen doménu. Výsledné URL by mělo být ve tvaru <domain>/rest/collect.

Původní spouštění testů.

Code 7.3: Spuštění pomocí Cucumber.class

```
@RunWith(Cucumber.class)
@CucumberOptions(features = {
                    myFeature.feature
        },
        glue = "mypackage")
public class MyRunnerClass{
}
```

Spouštění po úpravě runneru.

Code 7.4: Spuštění pomocí UnoCucumberRunner.class

```
@RunWith(UnoCucumberRunner.class)
@CucumberOptions(features = {
        myFeature.feature
        },
        glue = "mypackage")
public class MyRunnerClass{
}
```

Tato změna je potřeba, protože aktuální verze původního Cucumber runneru neposkytuje všechna potřebná data pro běh a je tedy potřeba si je opatřit. Po těchto úpravách a konfiguraci by mělo odesílání dat na server fungovat.

Před prvním použitím části aplikace určené k reprezentaci dat je nutné nakonfigurovat proměnné, které slouží pro vytváření odkazů. Momentálně je podporovaná pouze jedna proměnná a jedná se o URL, kde uživatel může nalézt logy z průběhu sestavení. Odkaz do konfigurační sekce je umístěn ve spodní části jakékoliv obrazovky v aplikaci určené pro prezentaci dat. Pro přístup je vyžadováno jméno o heslo.

Defaultní jméno: **admin** a heslo: **cucumber123** pro přístup do konfigurační sekce.

Kapitola 8

Ekonomicko-manažerské zhodnocení

Kapitola pojednávající o přínosech této práce z hlediska ekonomického alias jakou formou tato práce prospěje společnosti při provozování podnikání a manažerského, kde se zaměřím na přínosy práce při nutnosti činit rozhodnutí.

8.1 Náklady

Náklady na vývoj aplikací mají mnoho různých podob. Do nákladů tedy nestačí počítat pouze náklady spojené s fází implementační, ale je nutné zahrnout i analýzu, návrh a součinnost.

Celkové náklady jsem tedy počítal od úplného počátku. To znamená od chvíle, kdy vznikla myšlenka, že je potřeba najít řešení pro daný problém. Do nákladů se tím dostanou i odhady na pracnost spojenou s následným nastudováním problematiky a hledáním řešení mezi existujícími technologiemi.

Z existujících řešení žádné plně nepokrývalo dané požadavky. Proto bylo nutné provést důkladnější analýzu rámcových požadavků, návrh aplikace a výběr technologií pro následnou implementaci. To také zvýšilo finální náklady na vývoj.

V následující tabulce jsou uvedeny odhadované pracnosti za výše vyjmenované oblasti v MD.

Při průměrných nákladech 3 000 Kč na MD, pak odhadované celkové náklady na vývoj této aplikace jsou 273 000 Kč. Vývoj bude pokračovat i po dokončení prací v rozsahu diplomové práce, avšak to už je mimo rozsah této práce.

Pro koncového uživatele, pak celkové náklady tvoří, jen náklady spojené s provozem. Aplikace bude s největší pravděpodobností nasazena na interní infrastruktuře a proto do nákladů nemá smysl započítávat náklady za pronájem domény. Naopak započtu náklady spojené s administrací interní in-

Tabulka 8.1: Náklady na vývoj aplikace

Fáze	**MD**
Specifikace zadání	3
Analýza zadání	4
Zhodnocení existujících zadání	2
Návrh	4
Teoretická příprava na implementaci	5
Implementace	60
Součinnost ze strany test. týmu	3
Součinnost ze strany týmu pro CI	1
Součinnost ze strany SIT týmu	1
Testování a integrace	5
Prezentace výsledků	1
Vytížení Scrum mastera	2
Celkem	91

frastruktury a potřebou občasné konfigurace aplikace při změnách v interakci se spolupracujícími systémy.

Odhad průměrných nákladů na provoz serveru s aplikací: 5 000 Kč/rok. Odhad průměrných nákladů spojených s administrací cca 8 000 Kč/rok. Opět při průměrných nákladech 3 000 Kč za MD.

8.2 Přínosy

Přínosů, které použití aplikace přinese je hned několik.

8.2.1 Rychlejší reakce na spadlé testy

Jako jeden z cílů aplikace je usnadnit týmu testerů jejich práci. Aplikace se snaží poskytnout co nejvíce potřebných dat pro odhalování příčin neprůchozích testů, snaží se být uživatelsky přívětivá a testeři tak mohou rychle vyhodnotit, zda je chyba v testu, nebo zda se jedná o bug.

Splnění tohoto cíle tak přináší úsporu časovou při spouštění regresních testů v podobě rychlejší identifikace problému a i jeho následného řešení.

8.2.2 Statistiky

Statistické vyhodnocení a tendence chování testů v čase mohou napomoci se lépe připravit a začít predikovat, jaký má na aplikaci dopad například vydání nové verze aplikací třetích stran a kolik nových bugů asi vznikne po

nasazení větších změn nebo nasazení nové verze aplikace na produkční prostředí.

Přínosem je lepší predikce dopadů na aplikaci tím lepší odhady pracnosti spojené s opravami chyb a úpravami aplikace. To usnadní a hlavně zpřesní odhady potřebných zdrojů.

Zlepšení odhadů napomůže i ke správnějšímu určení termínů pro dokončení projektu a tím lze předejít stresovým situacím a tlaku z vyšších vrstev vedení na uspíšení prací.

8.2.3 Důkazy funkčnosti

Vlastník produktu v průběhu projektu stanovuje, co je potřeba implementovat. Požadavky na funkčnost se v průběhu vývoje mohou měnit. To znamená, že požadavky mohou přibývat i ubývat v případě, že ještě neproběhla implementace. Proto vlastník produktu často vyžaduje poskytnutí přehledu o již implementované funkcionalitě a to i jinou formou než ujištění. Funkční testy, které podkládají funkčnost implementovaných požadavků jsou proto velmi vhodnou formou doložení implementované funkcionality.

Vlastník produktu tak získává přehled o funkcionalitě, již implementované, nebo právě implementované. Pokud tedy testy obsahují jasnou informaci o tom, kdy jsou napsané.

Tímto způsobem společnost dosáhne úspory při přípravách reportů pro zákazníka. Zákazník si může ověřit pokrok na projektu (po zaškolení jak systém ovládat) sám a není nutné věnovat tolik prostředků na přípravu prezentací a další materiálů pro podložení funkcionality. Samozřejmě dema implementované funkcionality se konají stále.

Zákazník tak nemusí na ukázky postupu na projektu čekat a získá pocit, že projekt postupuje dle jeho představ. A spokojený zákazník znamená další zakázky.

8.2.4 Důkazy nefunkčnosti

Skupiny testů jsou obvykle ve správě jednoto týmu. Velké množství nefunkčních testů v jedné skupině testů, tak může indikovat problémy, které je třeba řešit jinak než jen jejich opravou.

8.3 Shrnutí

Přínosy vyjmenované v předchozí sekci jasně převyšují náklady spojené s provozem aplikace.

Závěr

Cílem práce bylo vytvořit systém, který podpoří potřeby spojené s prezentací dat z testů, které jsou vytvořené pomocí Cucumber frameworku a to pro skupinu uživatelů s širokým spektrem požadavků. Ke splnění tohoto cíle bylo nutné projít celý vývojový cyklus. Nejdříve proběhl sběr požadavků na funkcionalitu systému a jejich analýza následovaná průzkumem existujících řešení. Při průzkumu jsem došel k závěru, že existující řešení a to ani jejich kombinace nepostačí na pokrytí požadavků a tak mohla započít podrobnější analýza a návrh nového systému. Po prioritizaci požadavků a rozdělení projektu do několika menších implementačních fází mohla započít implementace, testování a integrace první části, která je popsaná v této práci. Práce po domluvě s projektovým manažerem bude dostupná v gitovém repozitáři a vedena jako open source projekt. Githubový repozitář bude udržován při životě a všechny budoucí rozšíření a implementace dalších fází budou jeho součástí. Pokud si čtenář přeje vyzkoušet aplikaci, pak práce obsahuje i kapitolu s návodem pro instalaci. Práce navíc obsahuje ekonomicko-manažerské zhodnocení přínosů práce a tak si čtenář může udělat obrázek o tom, jaké systém poskytuje přínosy.

To však je již vyjmenovávání dosažených cílů, kterých by nebylo možné dosáhnout bez dlouhé cesty. Na jejímž počátku existovalo jen několik požadavků a mlhavá představa o tom, jak by mělo výsledné řešení vypadat. V průběhu vývoje bylo potřeba překonat mnoho překážek, upravovat a rozšiřovat zadání z následujících důvodů. Za posledních několik měsíců se na našem projektu ustanovily nové praktiky a postupy pro psaní testů v Cucumber frameworku. Dále se změnil celý postup sestavování projektu a to hned několikrát, což ovlivnilo celkem podstatně některé části aplikace. Na projektu se začal formovat tým pro systémovou integraci, který od aplikace požaduje další funkcionalitu. Všechny tyto změny bylo nutné promítnout do všech vývojových fází projektu a s tím se navyšovala i potřebná pracnost.

Nicméně dle mého názoru se všechny potřebné kroky pro implementaci první fáze projektu podařily a aplikace má poměrně dobrou šanci stát se úspěš-

ným konkurentem stávajících řešení uvedených v sekci s existujícími řešeními. Hlavně však v sektoru větších projektů.

Jsem rád, že jsem se dostal k implementaci aplikace tohoto typu. Práce na tomto projektu mi přinesla zkušenosti v několika oblastech a určitě se budou v budoucnu hodit. Zaprvé jsem rozšířil své obzory v oblasti testování aplikací na různých úrovních a pochopil potřebu takového testování a to z jiného pohledu než jen psaním testů. To je podle mě důležitá zkušenost, protože si jinak lze jen těžko uvědomit, proč se testuje na několika úrovních a jak lze dobře napsané testy mohou pomoci. Zadruhé v posledních letech Cucumber framework získává na popularitě a používá nebo se používat začíná na velkém množství projektů napsaných v různých jazycích a reportovacích nástrojů pro tento framework není zas tak mnoho. Budu se tak i nadále snažit i po dopsání této práce rozvíjet tento projekt, jelikož v něm vidím poměrně velký potenciál pro zaplnění potřeby po dobrém reportovacím nástroji. Budu tak rád, pokud čtenáři této práce aplikaci vyzkouší a náměty na zlepšení uvedou jako issue do githubového repozitáře stengvac/cucumber-reports [4].

Děkuji za váš čas při čtení práce, snad vám informace zde uvedené budou přínosem.

Literatura

[1] Cucumber. [cit. 2017-04-02]. Dostupné z: https://cucumber.io/

[2] GitHub. [cit. 2017-04-09]. Dostupné z: https://github.com/

[3] community, J.: Jenkins. [cit. 2017-04-24]. Dostupné z: https://jenkins.io/

[4] Štengl, V.: stengvac/cucumber-reports. [cit. 2017-05-04]. Dostupné z: https://github.com/stengvac/cucumber-reports

[5] community, R.: Ruby Programming Language. [cit. 2017-04-24]. Dostupné z: https://www.ruby-lang.org/en/

[6] Documentation Cucumber. [cit. 2017-05-04]. Dostupné z: https://cucumber.io/docs

[7] java.com: Java + You. [cit. 2017-04-24]. Dostupné z: https://www.java.com/en/

[8] collective, C.: Gherkin · cucumber/cucumber Wiki. [cit. 2017-04-24]. Dostupné z: https://github.com/cucumber/cucumber/wiki/Gherkin

[9] Hellesøy, A.: Feature Introduction · cucumber/cucumber Wiki. [cit. 2017-05-04]. Dostupné z: https://github.com/cucumber/cucumber/wiki/Feature-Introduction

[10] Given When Then. [cit. 2017-04-03]. Dostupné z: https://github.com/cucumber/cucumber/wiki/Given-When-Then

[11] Enterprise Architect - UML Design Tools and UML CASE tools for software development. [cit. 2017-04-16]. Dostupné z: http://www.sparxsystems.com/products/ea/

[12] Welcome to Apache Maven. [cit. 2017-04-29]. Dostupné z: https://maven.apache.org/

[13] junit team: JUnit. [cit. 2017-04-11]. Dostupné z: http://junit.org/junit4/

[14] IntelliJ IDEA the Java IDE. [cit. 2017-04-25]. Dostupné z: https://www.jetbrains.com/idea/

[15] Hlava, T.: Úrovně provádění testů | Testování softwaru. 21.8.2011 [cit. 2015-03-29]. Dostupné z: http://testovanisoftwaru.cz/category/metodika-testovani/druhy-typy-a-kategorie-testu/

[16] Gerrit Code Review. [cit. 2017-04-22]. Dostupné z: https://www.gerritcodereview.com/

[17] Hlava, T.: Funkční Testy | Testování softwaru. [cit. 2017-04-16]. Dostupné z: http://testovanisoftwaru.cz/tag/funkcni-testy/

[18] Torvalds, L.: Git. [cit. 2017-04-21]. Dostupné z: https://git-scm.com/

[19] Badle, B. J. B. A. B. E. B. D. B. J. ., Samit: Selenium - Web Browser Automation. [cit. 2017-04-20]. Dostupné z: http://www.seleniumhq.org/

[20] Szczepanik, D.: HTML reports for Cucumber. [cit. 2017-04-09]. Dostupné z: https://github.com/damianszczepanik/cucumber-reporting

[21] Aslak Hellesøy, B. R. S. G., Oleg Sukhodolsky: stengvac/cucumber-reports. [cit. 2017-05-04]. Dostupné z: https://github.com/cucumber/gherkin2/blob/master/java/src/main/java/gherkin/formatter/JSONFormatter.java

[22] Szczepanik, D.: Cucumber Reports Plugin. [cit. 2017-04-30]. Dostupné z: https://wiki.jenkins-ci.org/display/JENKINS/Cucumber+Reports+Plugin

[23] Nord, J.: jenkinsci/cucumber-testresult-plugin. [cit. 2017-04-25]. Dostupné z: https://github.com/jenkinsci/cucumber-testresult-plugin

[24] Jenkins: Jenkins plugin for Cucumber-JVM performance reporting. [cit. 2017-04-25]. Dostupné z: https://github.com/jenkinsci/cucumber-performance-plugin

[25] Stefan Huber, D. F. K. B. M. H., Christian Köberl: Stores results of BDD tests with Cucumber-JVM in a database and provides reporting capabilities. [cit. 2017-04-20]. Dostupné z: https://github.com/porscheinformatik/cucumber-report-db

[26] MongoDB for GIANT Ideas | MongoDB. [cit. 2017-04-25]. Dostupné z: https://www.mongodb.com/

[27] Livadariu, B.: The project offers the possibility to build test reports using a bootstrap components, offering rendering support on a wide range of devices. [cit. 2017-04-20]. Dostupné z: https://github.com/web-innovate/bootstraped-multi-test-results-report

[28] Livadariu, B.: Feature Summary Report. [cit. 2017-04-20]. Dostupné z: https://web-innovate.github.io/cucumber-reports/feature-reports/9cd26f45-2dc3-4294-8b94-dbd18d03404c.html

[29] The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR 303. [cit. 2017-04-25]. Dostupné z: https://jcp.org/en/jsr/detail?id=303

[30] Jetty - Servlet Engine and Http Server. [cit. 2017-04-30]. Dostupné z: http://www.eclipse.org/jetty/

[31] Continuous Code Quality | SonarQube. [cit. 2017-04-15]. Dostupné z: https://www.sonarqube.org/

[32] SonarJava Rules | SonarSource. [cit. 2017-05-04]. Dostupné z: https://www.sonarsource.com/why-us/products/codeanalyzers/sonarjava/rules.html

[33] Java bean mappings, the easy way! [cit. 2017-04-29]. Dostupné z: http://mapstruct.org/

[34] Apache Log4j 2 - Apache Log4j 2. [cit. 2017-04-29]. Dostupné z: https://logging.apache.org/log4j/2.x/

[35] Saloranta, T.: FasterXML/jackson. [cit. 2017-04-10]. Dostupné z: https://github.com/FasterXML/jackson

[36] Tatu Saloranta, P. G. J. R., Adam Stroud: jackson-databind/ObjectMapper.java at master FasterXML/jackson-databind. [cit. 2017-05-05]. Dostupné z: https://github.com/FasterXML/jackson-databind/blob/master/src/main/java/com/fasterxml/jackson/databind/ObjectMapper.java

[37] The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR 310. [cit. 2017-04-29]. Dostupné z: https://jcp.org/en/jsr/detail?id=310

[38] Saloranta, T.: Java Platform, Enterprise Edition (Java EE) 7. [cit. 2017-04-10]. Dostupné z: https://docs.oracle.com/javaee/7/

[39] Pete Muir, ., Antoine Sabot-Durand: Contexts and Dependency Injection for Java Specification | Contexts and Dependency Injection. [cit. 2017-04-29]. Dostupné z: http://www.cdi-spec.org/

[40] Spring. [cit. 2017-04-29]. Dostupné z: https://spring.io/

[41] Spring Data. [cit. 2017-04-29]. Dostupné z: http://projects.spring.io/spring-data/

[42] Fowler, M.: InversionOfControl. [cit. 2017-05-05]. Dostupné z: https://martinfowler.com/bliki/InversionOfControl.html

[43] Allen, D.: JavaServer Faces.org. [cit. 2017-04-29]. Dostupné z: http://www.javaserverfaces.org/

[44] AngularJS — Superheroic JavaScript MVW Framework. [cit. 2017-04-29]. Dostupné z: https://angularjs.org/

[45] A JavaScript library for building user interfaces - React. [cit. 2017-04-29]. Dostupné z: https://facebook.github.io/react/

[46] Bauke Scholtz, A. T.: OmniFaces Showcase. [cit. 2017-04-29]. Dostupné z: http://omnifaces.org/

[47] Home | Apache Wicket. [cit. 2017-04-29]. Dostupné z: https://wicket.apache.org/

[48] I banku můžete mít rádi • Air Bank. [cit. 2017-04-29]. Dostupné z: https://www.airbank.cz/

[49] TypeScript - JavaScript that scales. [cit. 2017-04-29]. Dostupné z: https://www.typescriptlang.org/

[50] PrimeTek: PrimeFaces | Ultimate UI Framework for Java EE. [cit. 2017-04-11]. Dostupné z: https://www.primefaces.org/

[51] Google Core Libraries for Java. [cit. 2017-04-11]. Dostupné z: https://github.com/google/guava

[52] Hibernate Validator - Hibernate Validator. [cit. 2017-04-29]. Dostupné z: http://hibernate.org/validator/

[53] junit team: Mockito framework site. [cit. 2017-04-11]. Dostupné z: http://site.mockito.org/

[54] Home - Scrum Alliance. [cit. 2017-04-29]. Dostupné z: https://www.scrumalliance.org/

[55] Hellesøy, A.: gherkin2/Reporter.java at master · cucumber/gherkin2. [cit. 2017-05-04]. Dostupné z: https://github.com/cucumber/gherkin2/blob/master/java/src/main/java/gherkin/formatter/Reporter.java

[56] Aslak Hellesøy, M. W., Sebastian Gröbler: gherkin2/Formatter.java at master · cucumber/gherkin2. [cit. 2017-05-04]. Dostupné z: https://github.com/cucumber/gherkin2/blob/master/java/src/main/java/gherkin/formatter/Formatter.java

[57] Balsamiq. Rapid, effective and fun wireframing software. | Balsamiq. [cit. 2017-04-29]. Dostupné z: http://balsamiq.com/

[58] Mark Otto, j.: Bootstrap; The world's most popular mobile-first and responsive front-end framework. [cit. 2017-04-30]. Dostupné z: http://getbootstrap.com/

[59] Apache Tomcat; - Apache Tomcat 8 Software Downloads. [cit. 2017-04-08]. Dostupné z: https://tomcat.apache.org/download-80.cgi

[60] Lightshot — screenshot tool for Mac and Win. [cit. 2017-05-02]. Dostupné z: https://app.prntscr.com/en/index.html

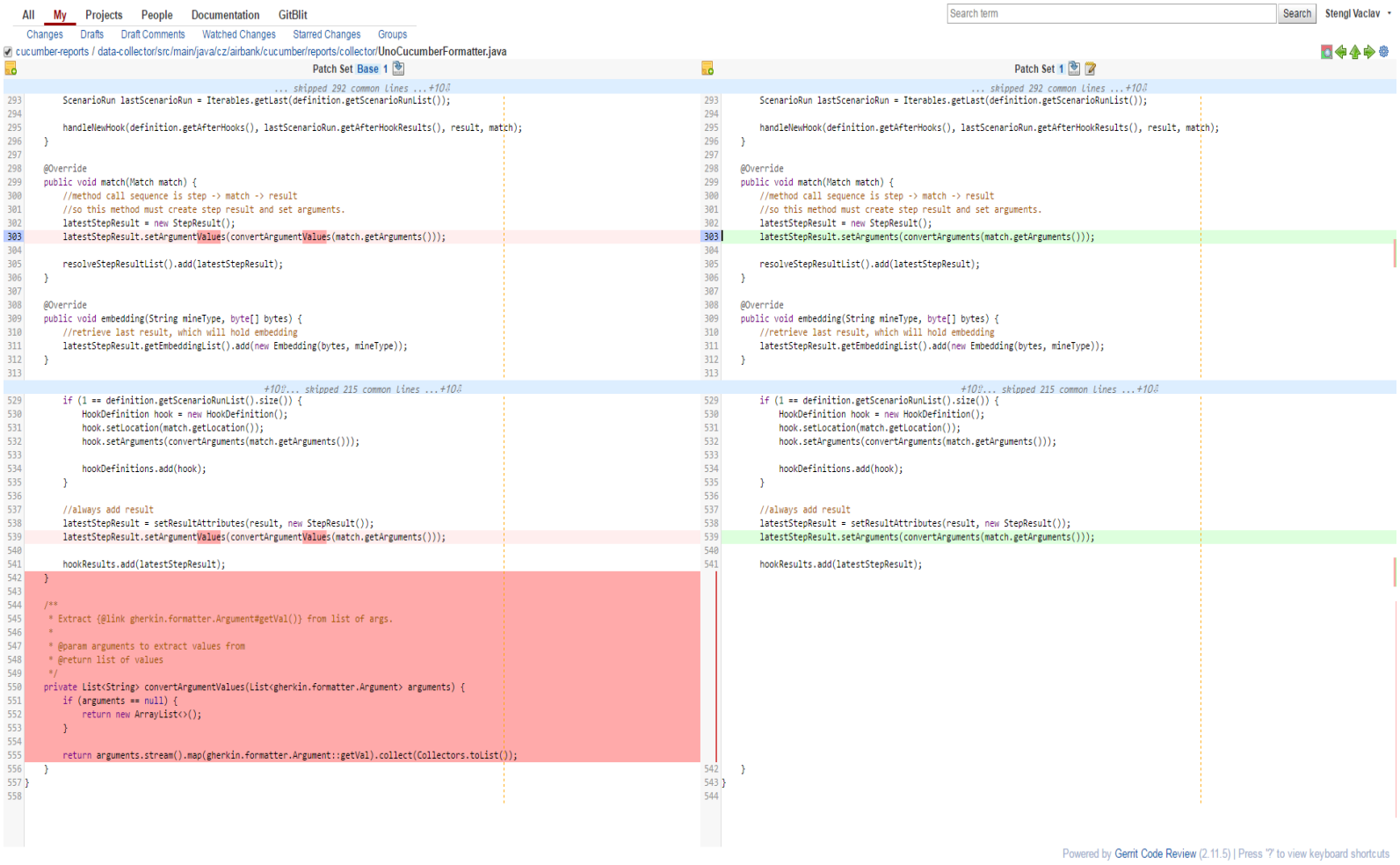
[61] Nord, J.: Cucumber Test Result Plugin. [cit. 2017-04-25]. Dostupné z: https://wiki.jenkins-ci.org/pages/viewpage.action?pageId=69764760&navigatingVersions=true

[62] Stefan Huber, D. F. K. B. M. H., Christian Köberl: Home porscheinformatik/cucumber-report-db Wiki. [cit. 2017-04-20]. Dostupné z: https://github.com/porscheinformatik/cucumber-report-db/wiki

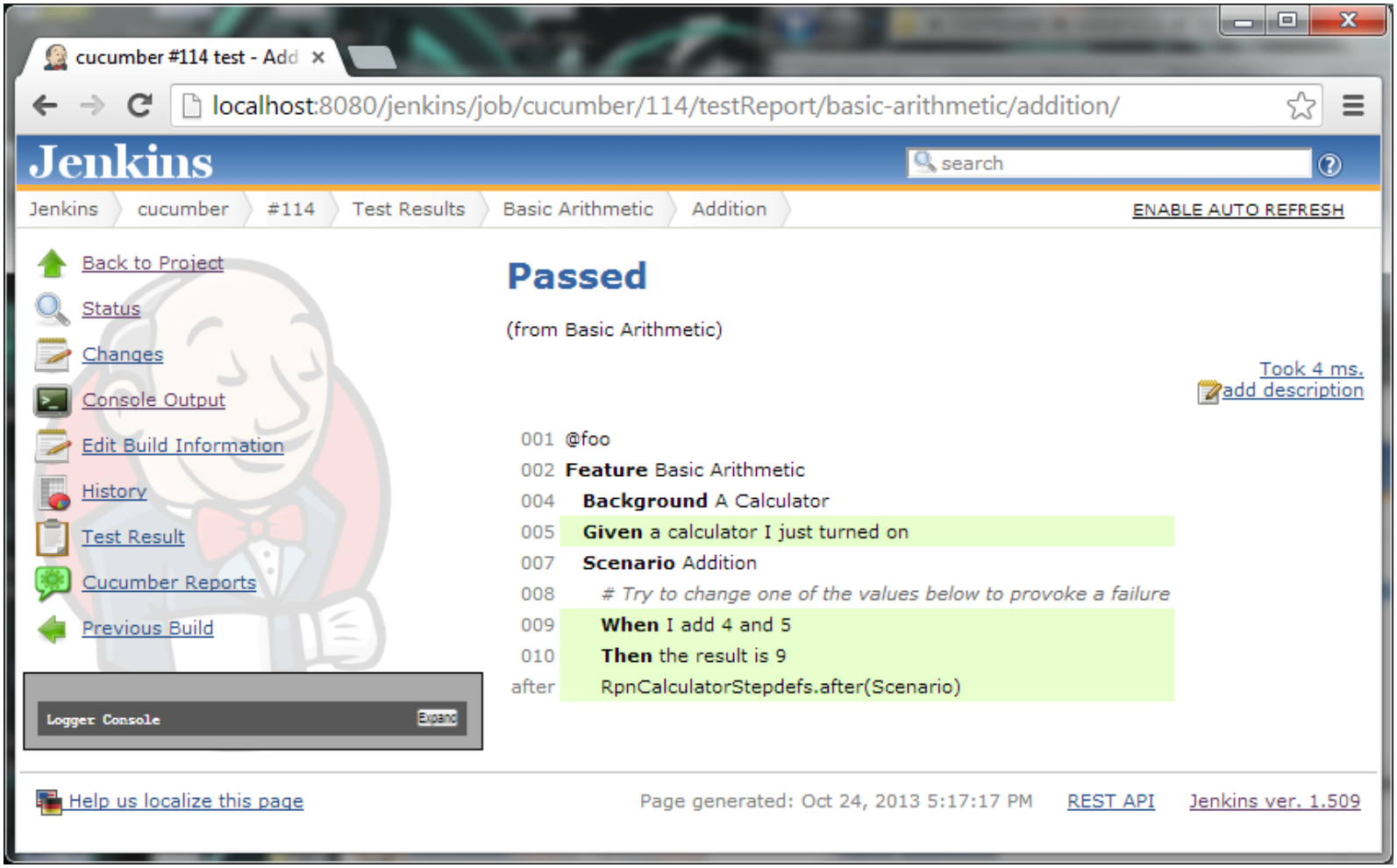
[63] Alder, G.: draw.io. [cit. 2017-04-30]. Dostupné z: https://www.draw.io/

Příloha A

Přílohy



Obrázek A.1: Gerrit - ukázka stránky pro code review. Vytvořeno pomocí [60]



Obrázek A.2: Cucumber reporting Jenkins - odkaz z Jenkins do reportů [61]

Project	Number	Date
Damian - project with Cucumber	10	11 gru 2016, 00:16

Branch	release/1.0
Browser	Firefox
Platform	Windows

Features Statistics

The following graphs show passing and failing statistics for features

	Steps						Scenarios			Features	
Feature	**Passed**	**Failed**	**Skipped**	**Pending**	**Undefined**	**Total**	**Passed**	**Failed**	**Total**	**Duration**	**Status**
1st feature	10	0	0	0	0	10	1	0	1	1m 39s 263ms	Passed
Second feature	5	1	2	1	2	11	1	1	2	092ms	Failed
2	**15**	**1**	**2**	**1**	**2**	**21**	**2**	**1**	**3**	**1m 39s 355ms**	
	71.43%	**4.76%**	**9.52%**	**4.76%**	**9.52%**		**66.67%**	**33.33%**			**50.00%**

Obrázek A.3: Cucumber reporting - přehled feature souborů v sestavení s výsledky [20]

Feature Report

	Steps						Scenarios			Features	
Feature	**Passed**	**Failed**	**Skipped**	**Pending**	**Undefined**	**Total**	**Passed**	**Failed**	**Total**	**Duration**	**Status**
Second feature	5	1	2	1	2	11	1	1	2	092ms	Failed

Feature Second feature

As an Account Holder I want to withdraw cash from an ATM,
so that I can get money when the bank is closed

@checkout

Scenario Outline Account may not have sufficient funds

Account holder withdraws more cash

Hooks

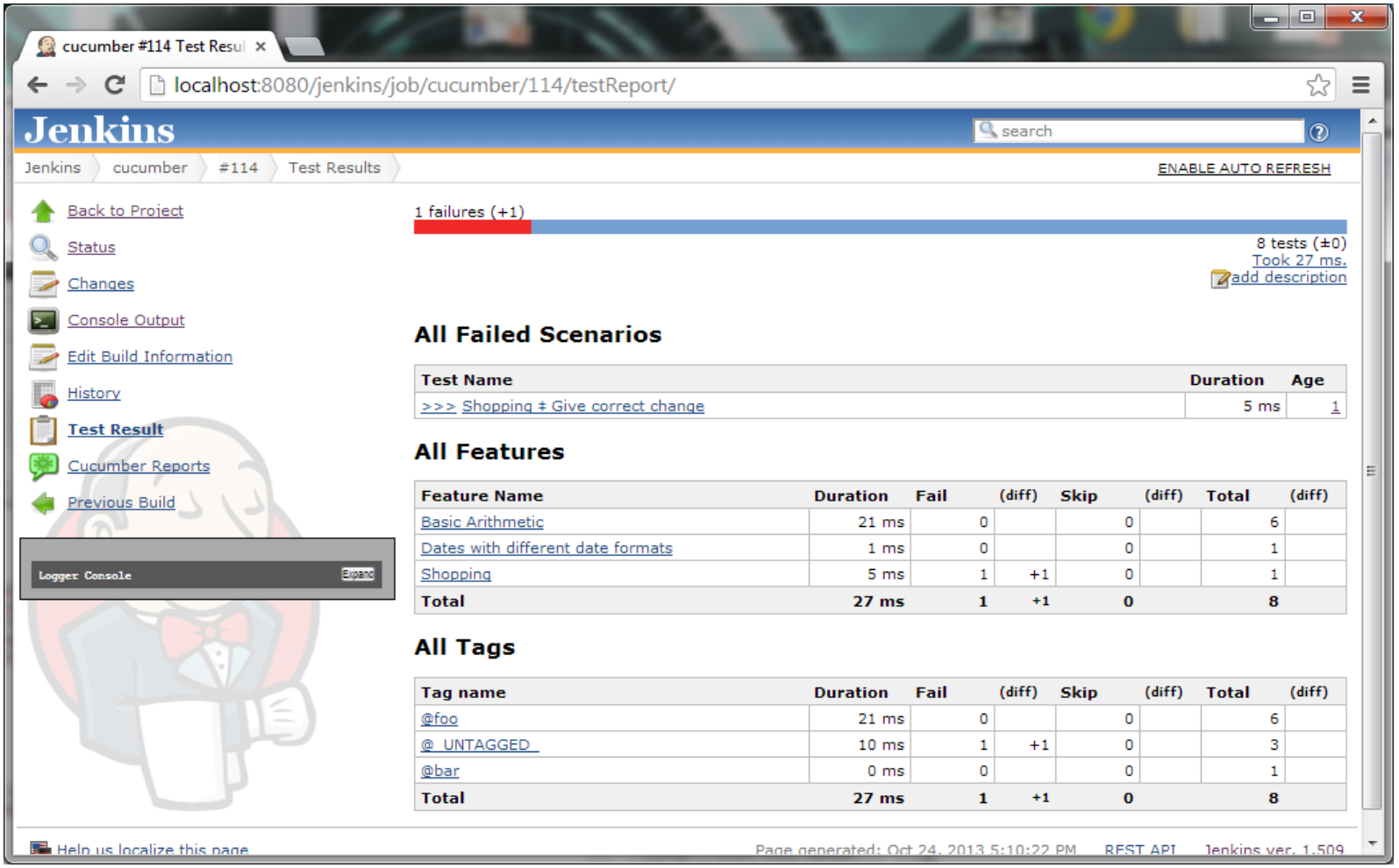
Before `MachineFactory.findCachMachine()`	010ms
Before `MachineFactory.wait()`	001ms

Steps

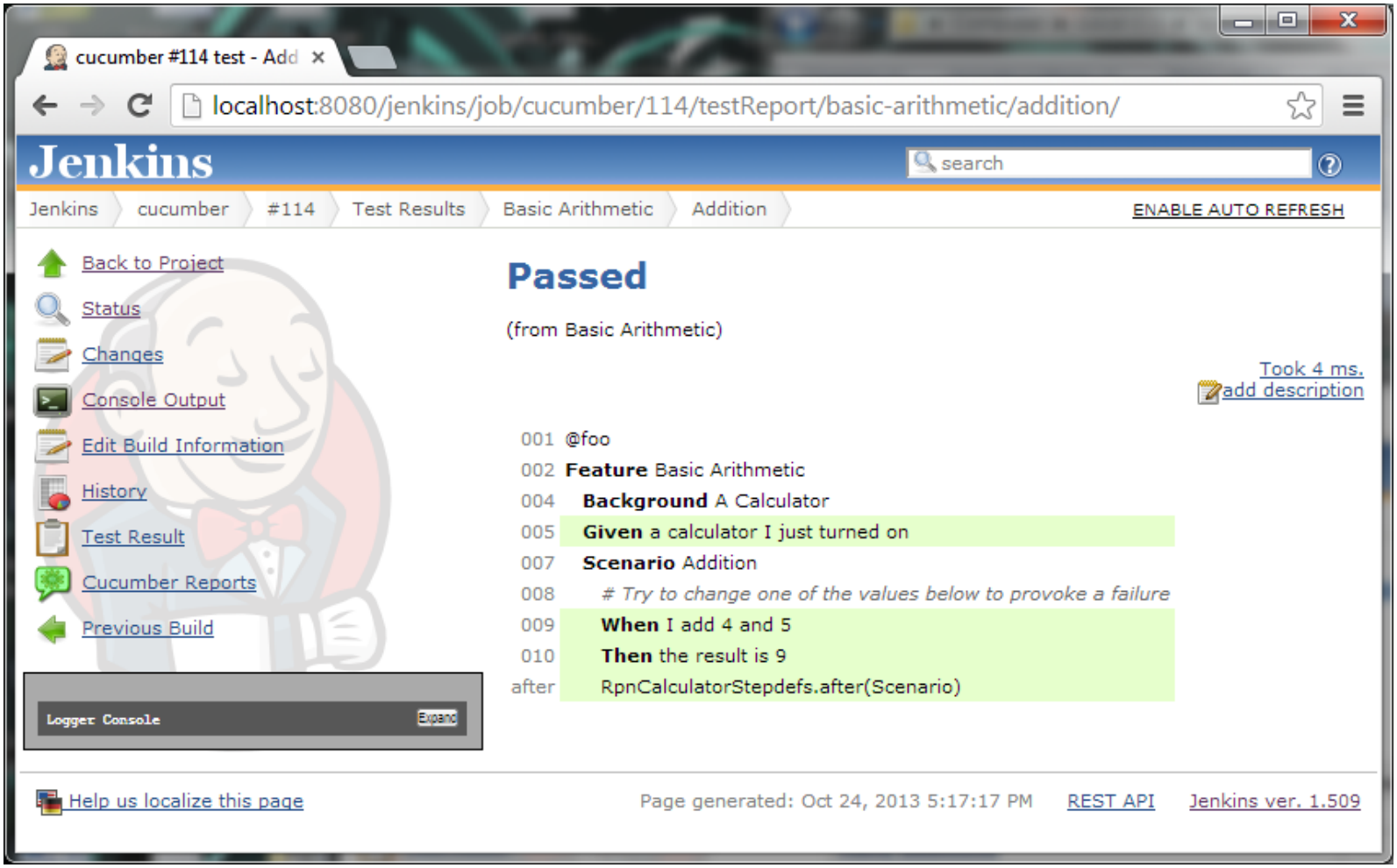
Given the account balance is 100	000ms
And the card is valid	000ms
And the machine contains 100	000ms
When the Account Holder requests 20	000ms
Then the ATM should dispense 20	000ms
And the account balance should be 90	001ms

Error message

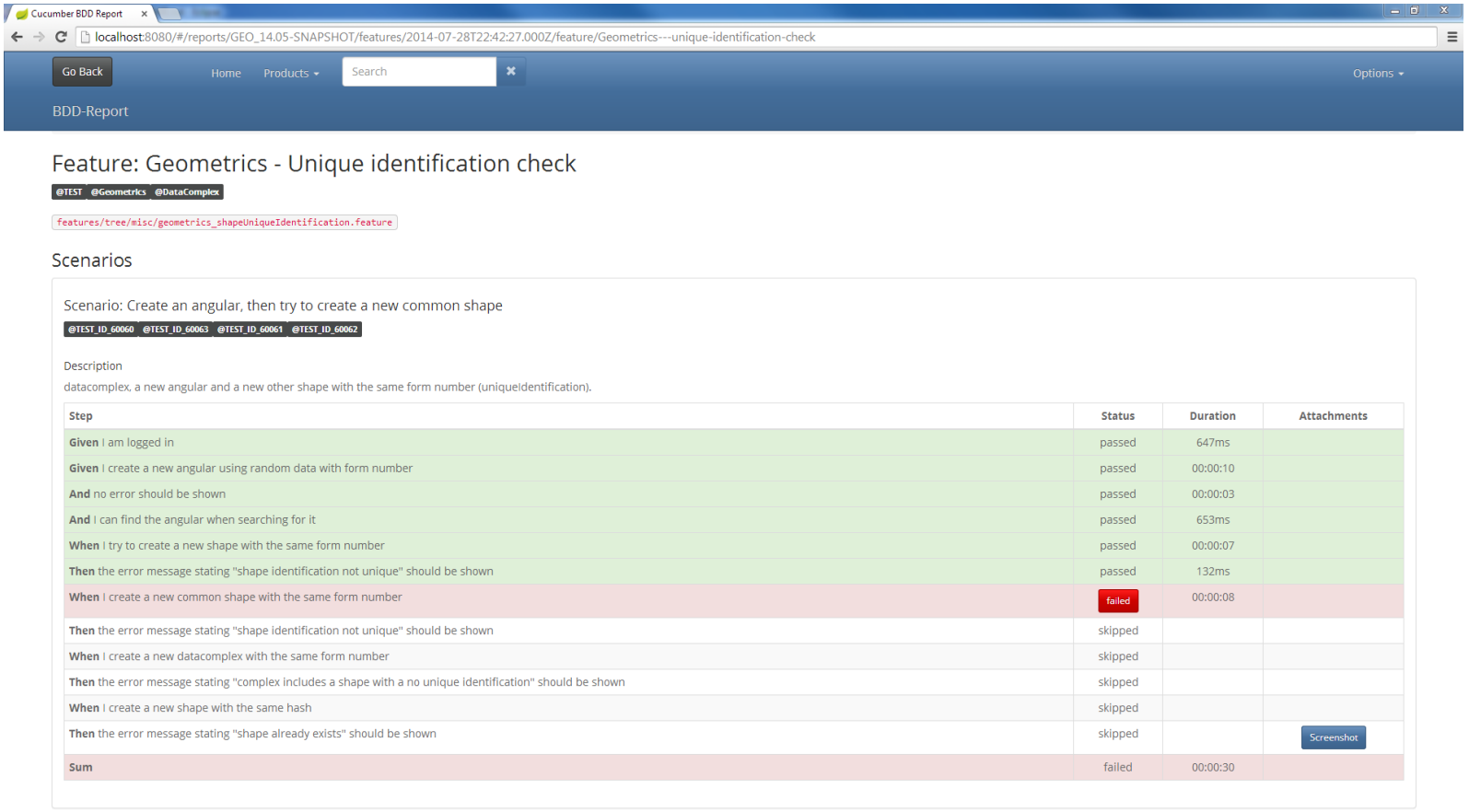
Obrázek A.4: Cucumber reporting - stránka s reportem feature souboru [20]



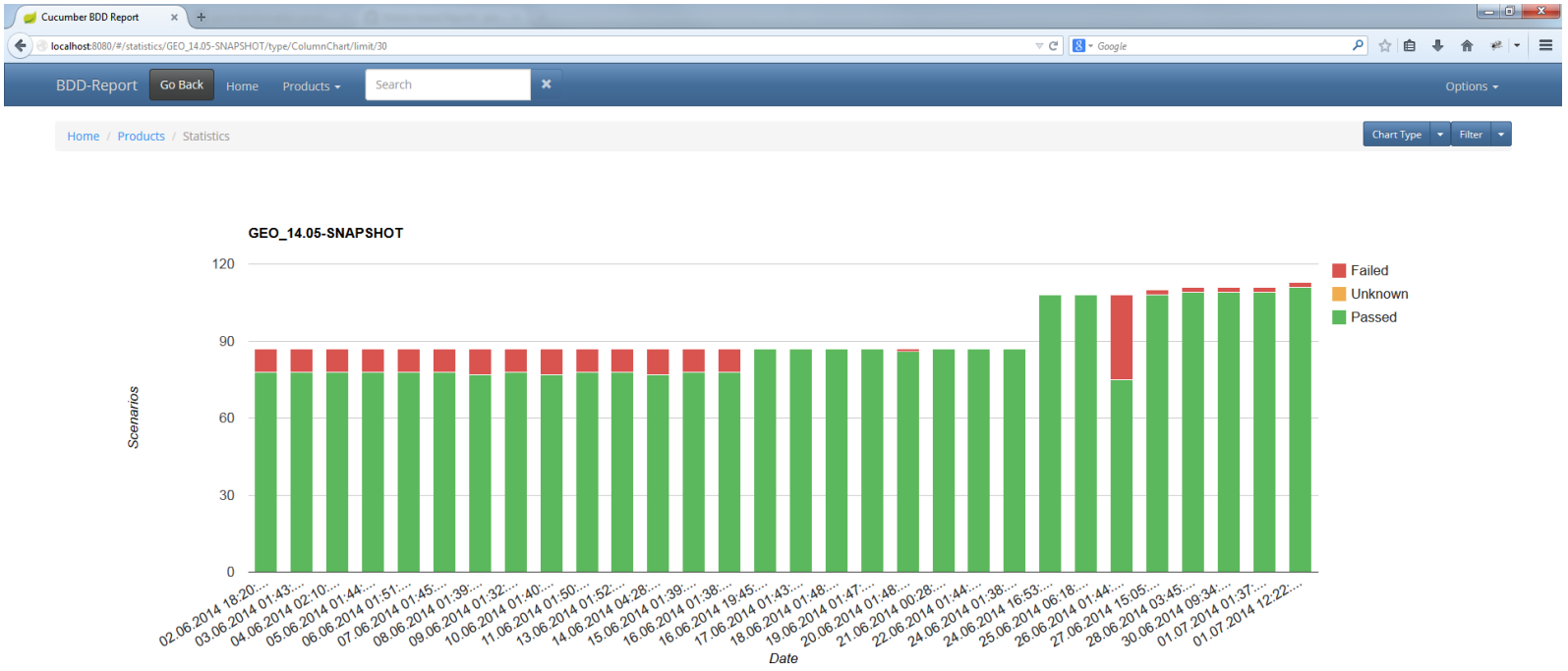
Obrázek A.5: Cucumber test result plugin - souhrnná stránka s výsledky feature souborů a tagů [?]



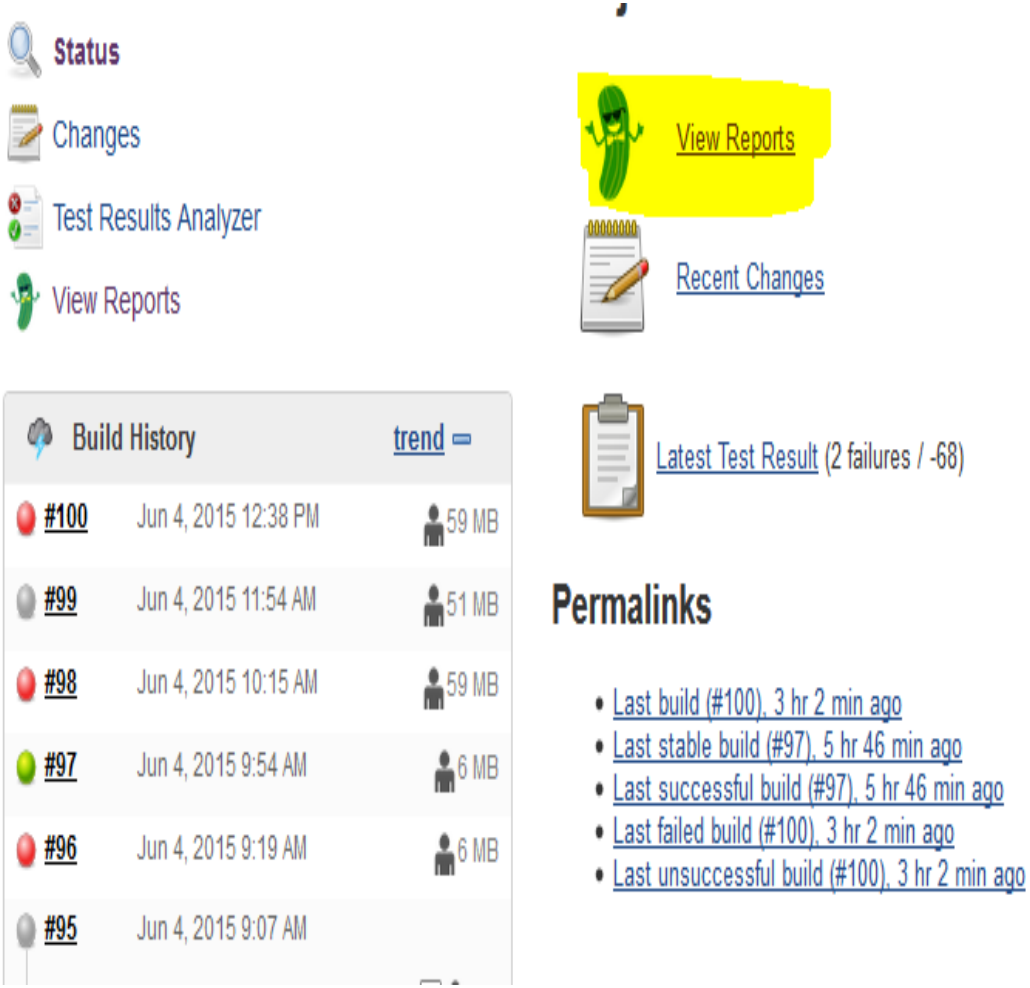
Obrázek A.6: Cucumber test result plugin - stránka s reportem feature souboru [?]



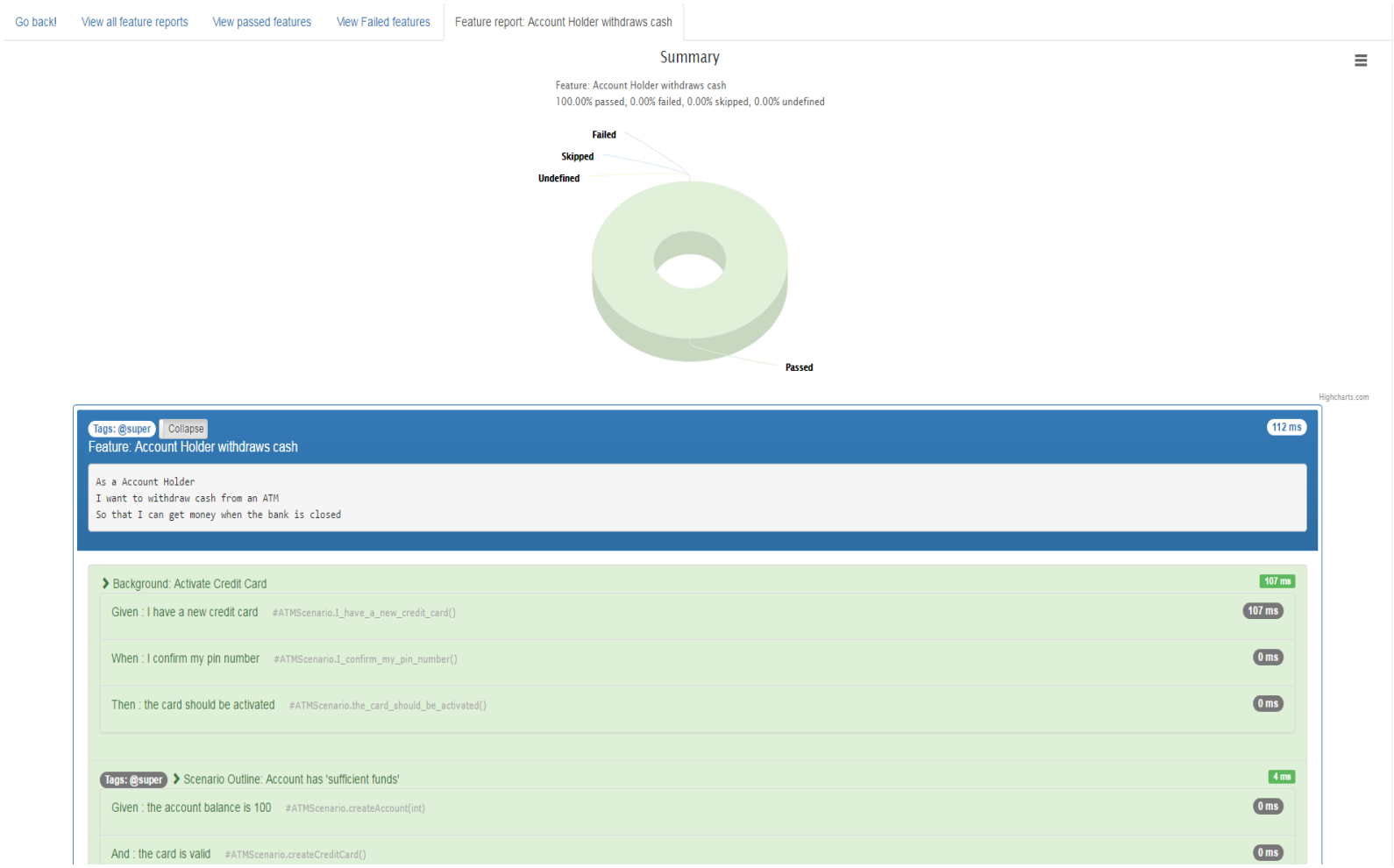
Obrázek A.7: Cucumber report DB - stránka s výsledkem běhu feature souboru [62]



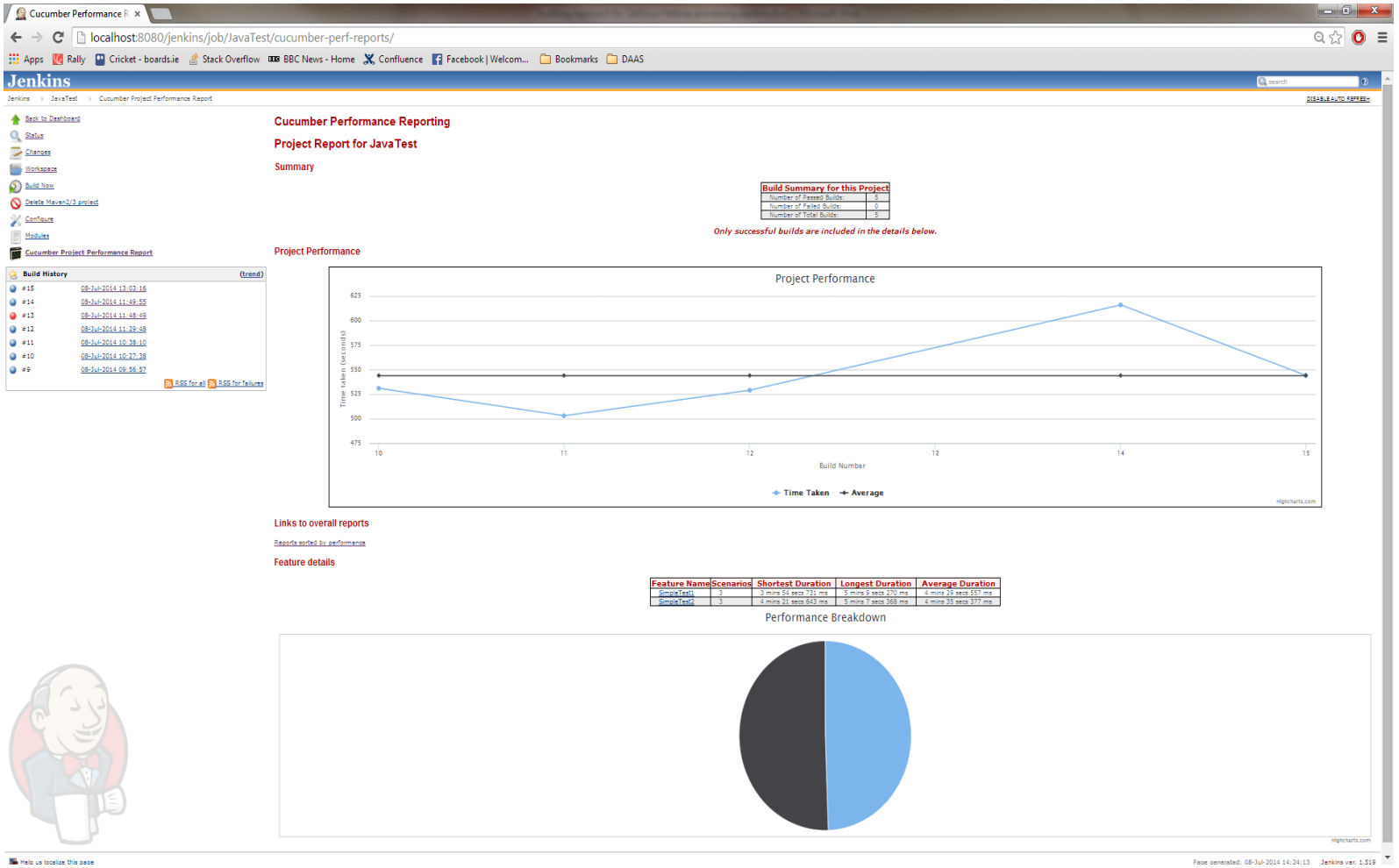
Obrázek A.8: Cucumber report DB - stránka zobrazující počet prošlých/padlých/neznámých scénářů v čase [62]



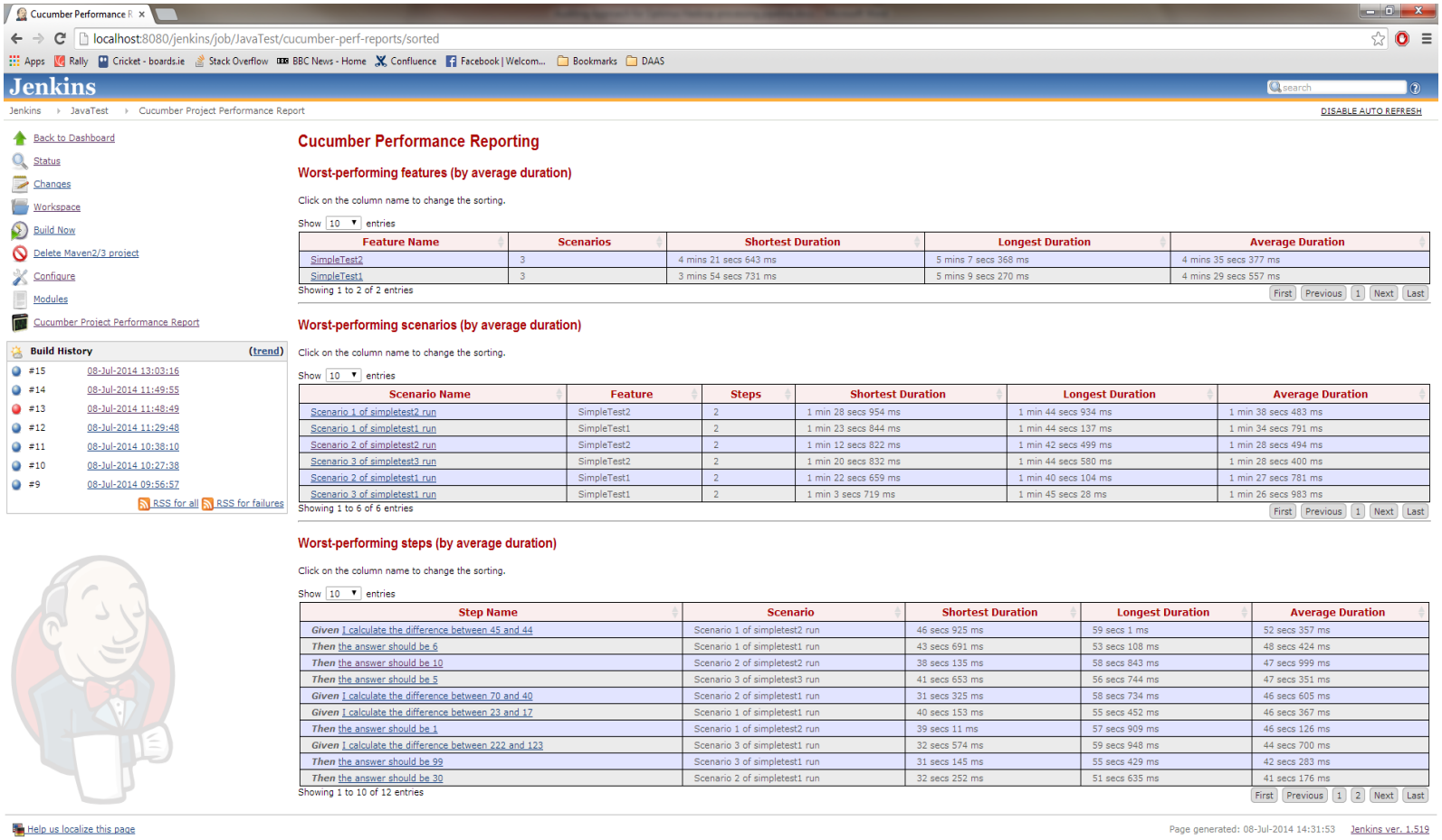
Obrázek A.9: Bootstraped Multi Test Results Report - vytvořená ikonka v Jenkinsu [27]



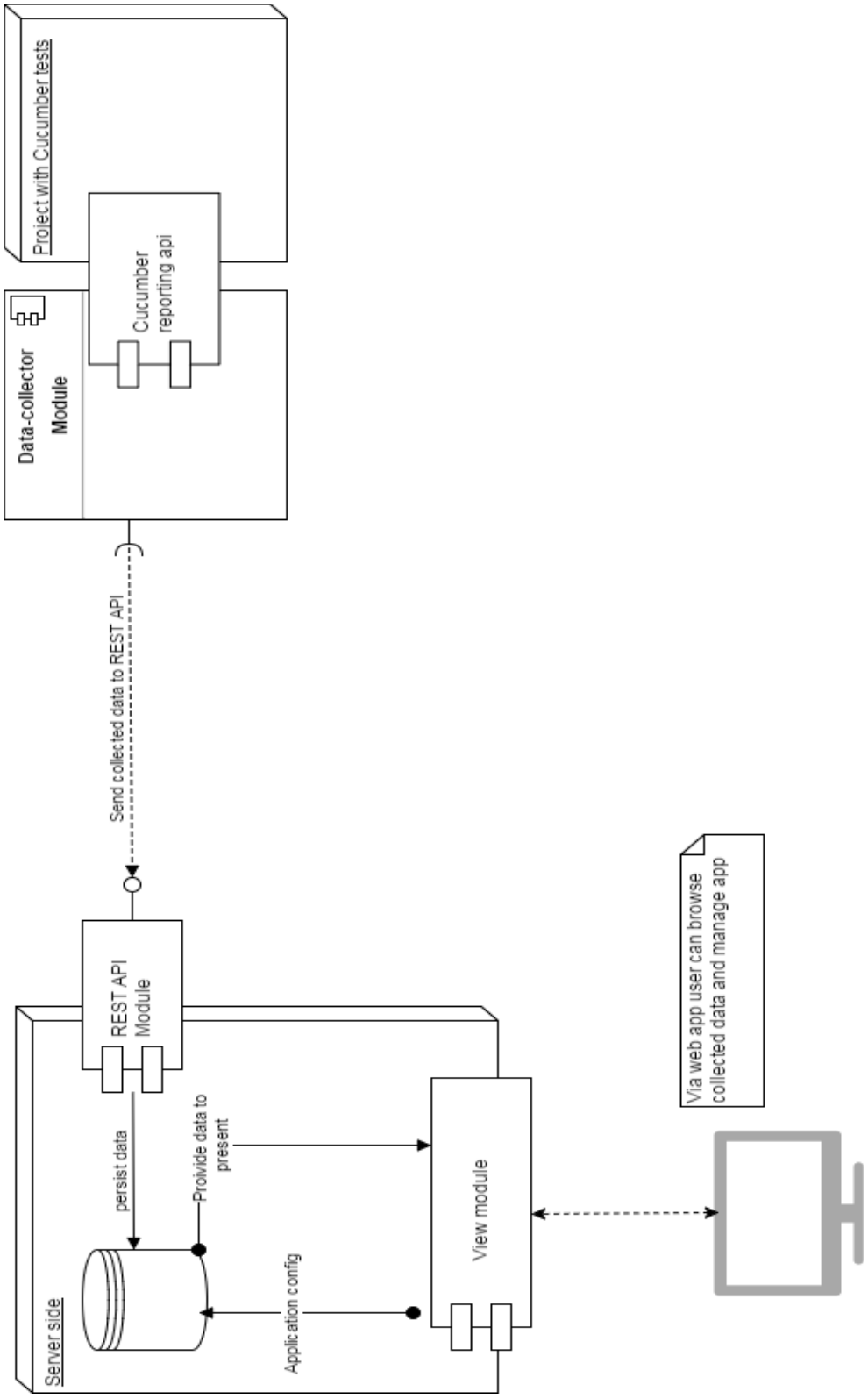
Obrázek A.10: Bootstraped Multi Test Results Report - stránka s reportem feature souboru. Vytvořeno pomocí [60] z [28]



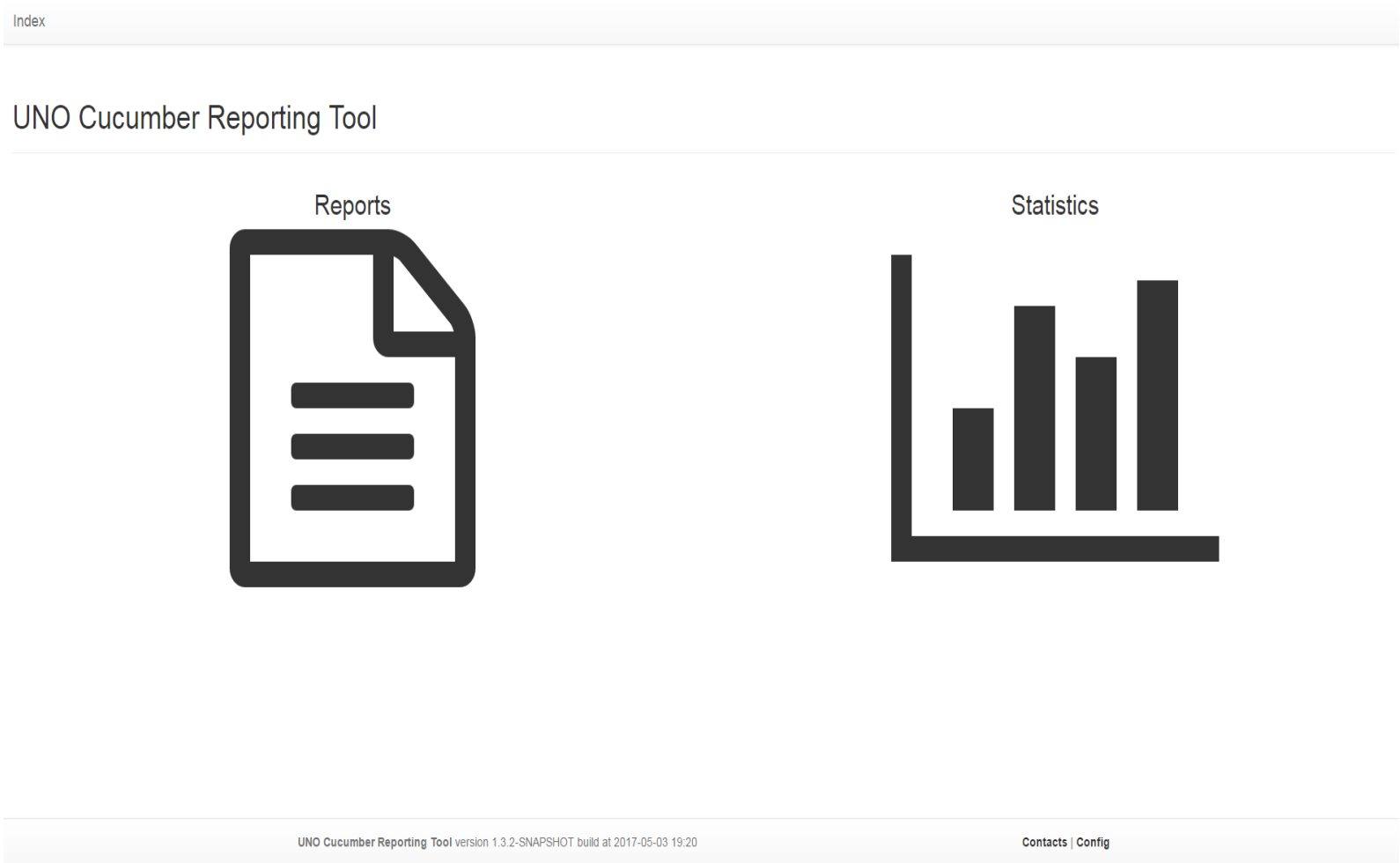
Obrázek A.11: Cucumber Performance Reports Plugin - stránka s vývojem trvání spuštění scénářů v čase [24]



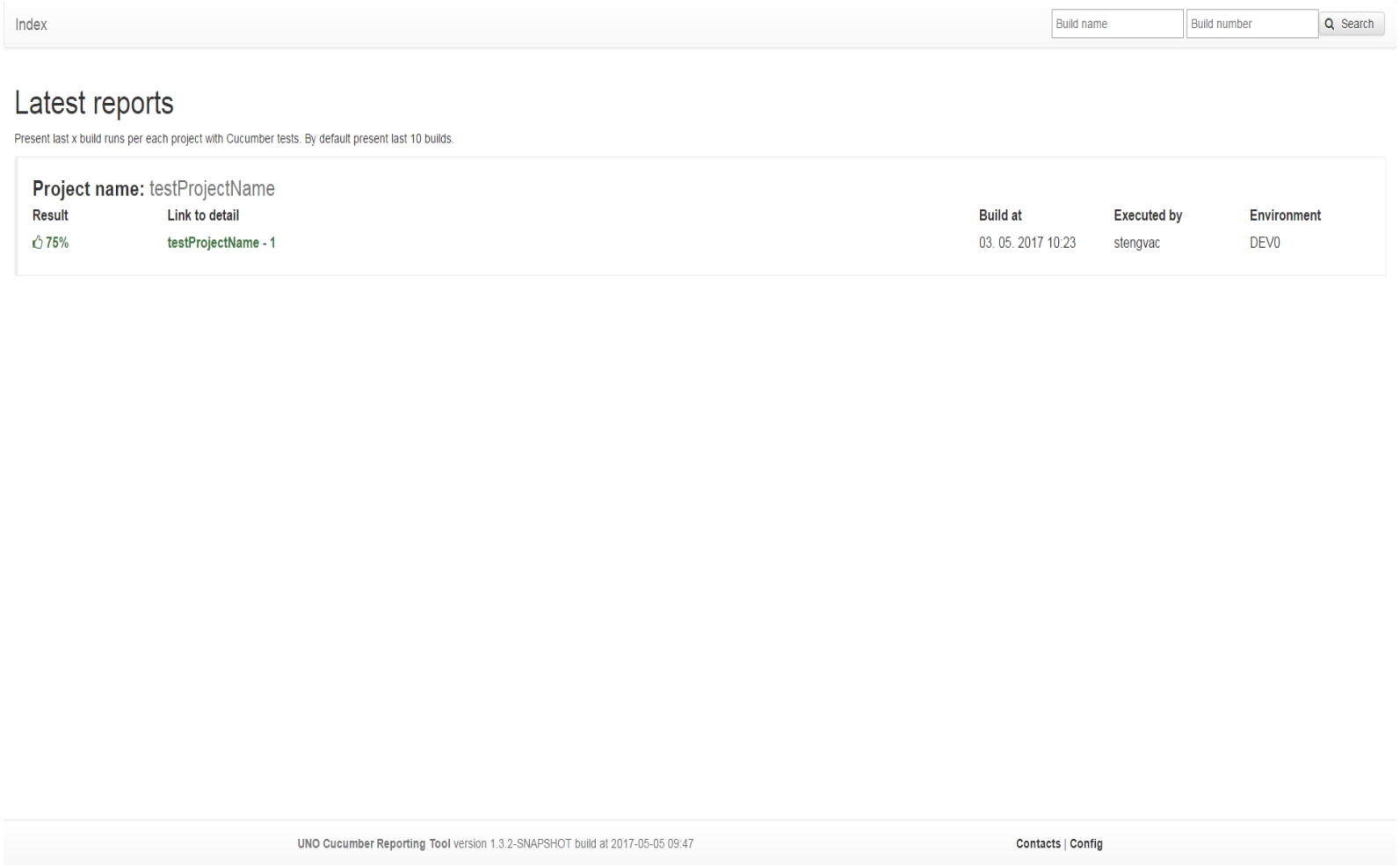
Obrázek A.12: Stránka s nejhoršími časy pro vlastnost, scénář, krok [24]



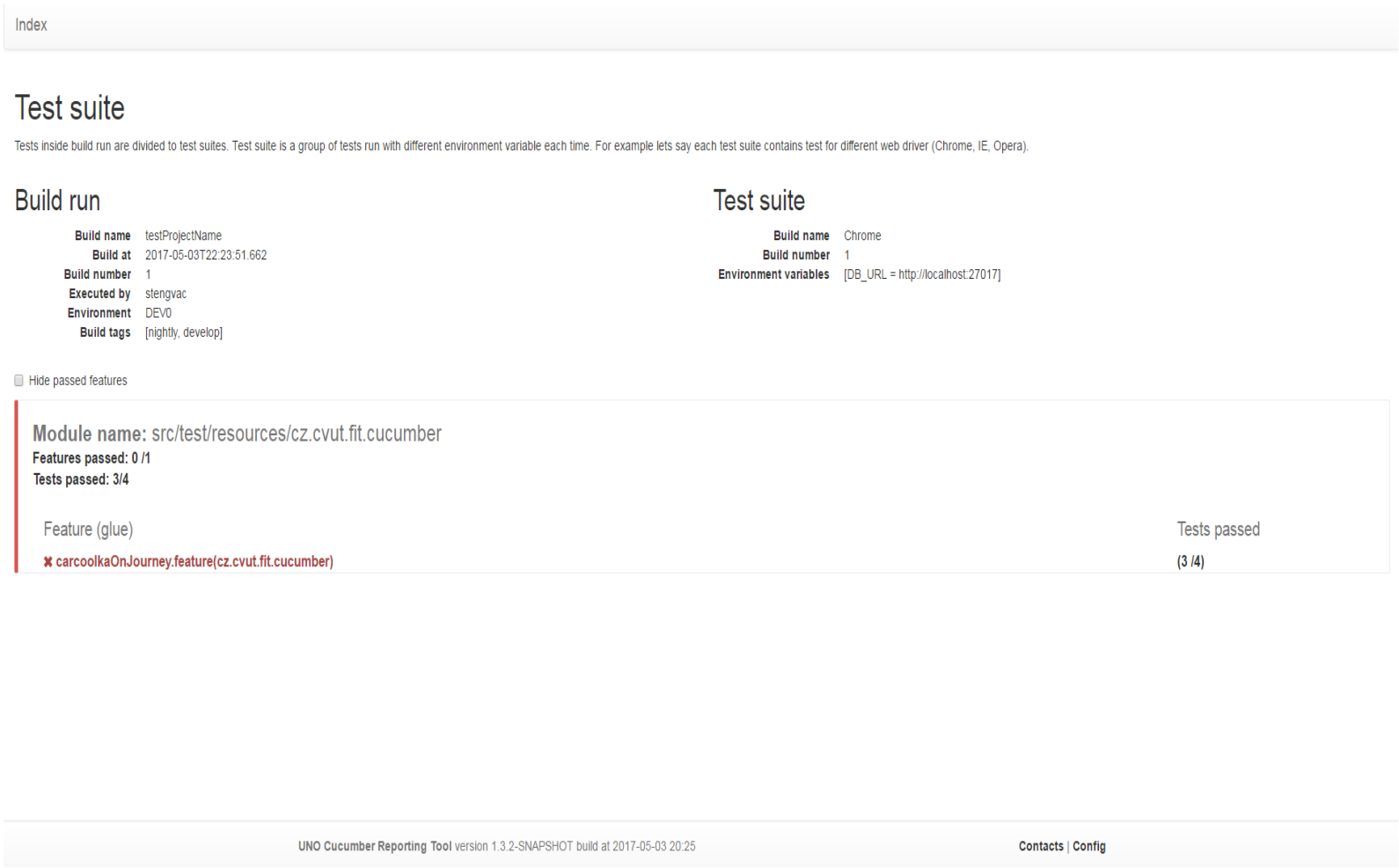
Obrázek A.13: Návrh aplikační architektury vytvořeno pomocí [63]



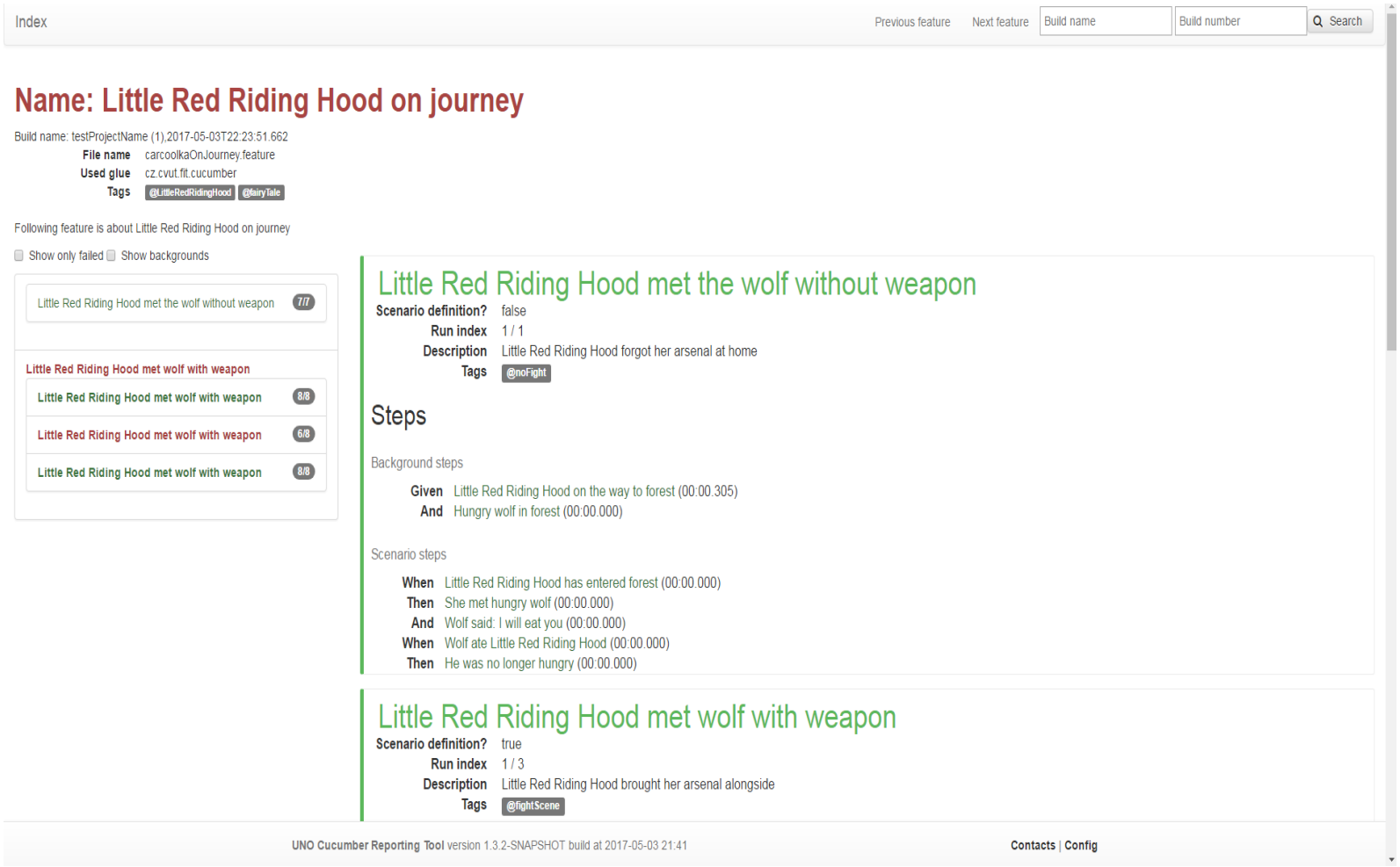
Obrázek A.14: Úvodní stránka prezentační vrstvy. Pořízeno pomocí [60]



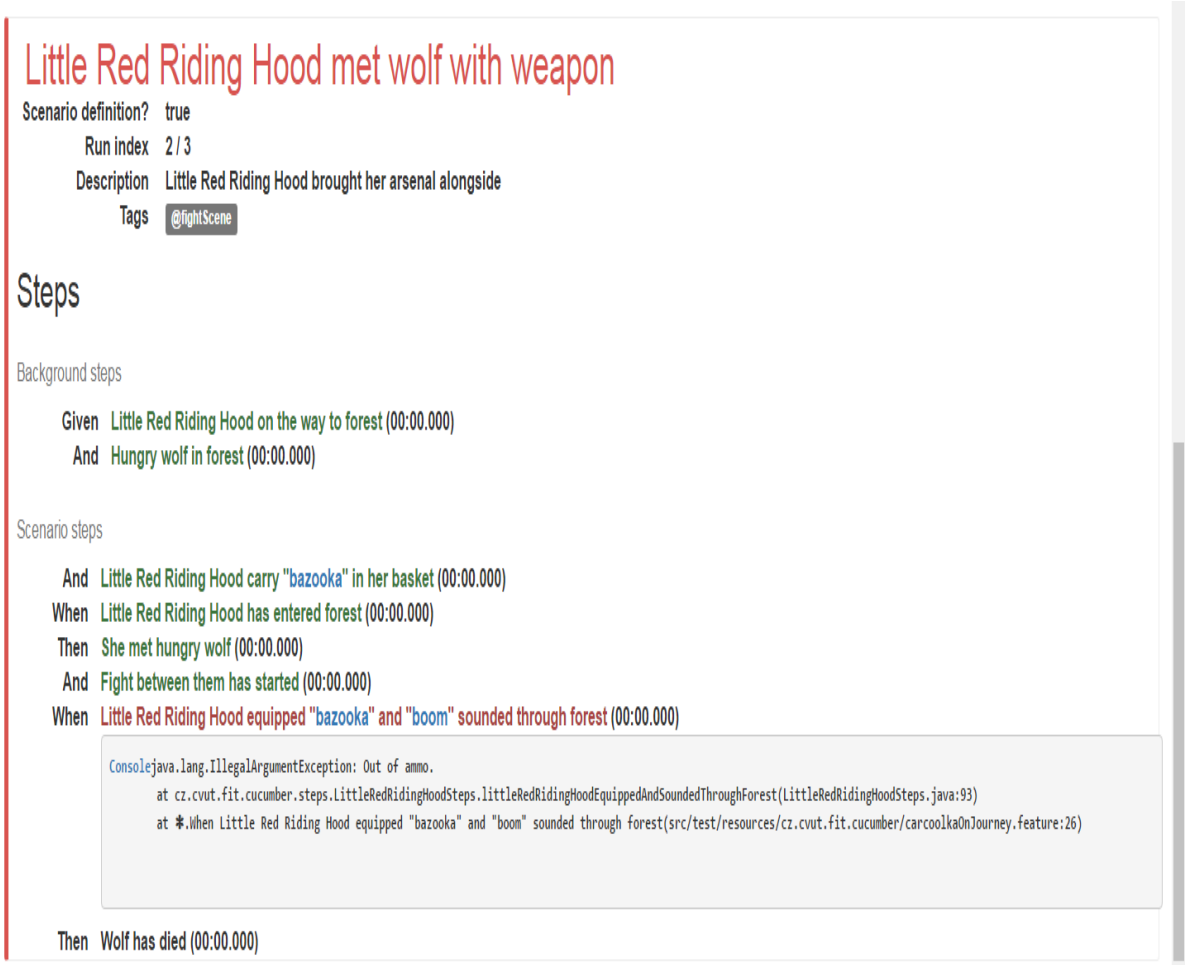
Obrázek A.15: Stránka s nejnovějšími sestaveními pro jednotlivé projekty. Pořízeno pomocí [60]



Obrázek A.16: Stránka s detailem sady testů. Pořízeno pomocí [60]



Obrázek A.17: Stránka s reportem feature souboru 1. Pořízeno pomocí [60]



Obrázek A.18: Stránka s reportem feature souboru 2. Pořízeno pomocí [60]

Příloha B

Seznam použitých zkratek

AOP Aspect Oriented Programming

API Application Programming Interface

BDD Behavioral Driven Development

CI Continuous Integration

DAO Data Access Object

DB Database

DI Dependency injection

GUI Graphical user interface

HW Hardware

ID Identifier

JSF Java Server Faces

MD Man day

POJO Plain Old Java Object

SW Software

TO Transport Object

UAT User Acceptance Testing

WAR Web Archive

XML Extensible markup language

Příloha C

Slovníček

sestavení označení pro proces, při kterém dochází ke kompilaci (pokud je třeba), generování potřebných zdrojů, spouštění testů

feature soubor soubor s koncovkou .feature, který Cucumber framework používá pro zápis testů v syntaxi specifikované jazykem Grerkin [8]. Testy v tomto souboru by měly být psané 'lidskou' řečí.

CI nástroj slouží pro průběžné sestavování zdrojového kódu vyvíjených projektů. Obvykle se jedná o automatizovaný proces, který se spustí při nahrání nového kódu do centrálního repozitáře. Tedy provede sestavení a tím poskytne základní informace o stavu kódu a nevyhovující kód je pak vrácen k přepracování.

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `imgs` ..... použité obrázky
    - `features` složka s feature souborem použitým pro ukázky aplikace
  - `cucumber-reports` ..... zdrojové kódy implementace
    - `common` ..... modul se sd9lenou funkcionalitou
    - `dao` ..... DAO modul
    - `data-collector` ..... modul pro sběr dat
    - `rest-api` ..... REST API modul
    - `transport-modelmodul s modelem pro prenos dat`
    - `viewmodul s prezentační vrstvou`
    - `pom.xml`
    - `README.md`
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF