VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

INTEGRACE NÁSTROJŮ S AUTOMATIZOVANÝM TESTOVÁNÍM GUI MOBILNÍCH APLIKACÍ

INTEGRATION OF TOOLS FOR AUTOMATED GUI TESTING OF MOBILE APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **RICHARD STEHLÍK**
AUTHOR

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

BRNO 2018

Abstrakt

Cílem práce je implementace automatizovaných testů uživatelského prostředí mobilních aplikací a jejich integrace do procesu průběžné integrace. Analýzou dostupných a běžně používaných nástrojů pro automatizaci vybereme vhodné kandidáty pro integrovaný systém, který má za cíl usnadnit, urychlit a zefektivnit vývoj v agilním prostředí.

Abstract

Goal of this thesis is implementation of automated UI tests for mobile applications and its integration into continuous integration process. With analysis of available and commonly used supporting software for automation, we will choose best candidates for integrated system, which goal is to make developement in agile environmet easier, faster and efective.

Klíčová slova

automatizace testování, testování GUI, mobilní testování, kontinuální integrace, integrace nástrojů

Keywords

test automation, GUI testing, mobile testing, continuous integration, tools integration

Citace

STEHLÍK, Richard. *Integrace nástrojů s automatizovaným testováním GUI mobilních aplikací.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

Integrace nástrojů s automatizovaným testováním GUI mobilních aplikací

Prohlášení

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

....
Richard Stehlík
16. května 2018

Poděkování

Děkuji vedoucímu své práce Ing. Aleši Smrčkovi, Ph.D. za cenné rady v průběhu psaní bakalářské práce. Dále bych rád poděkoval Františce Švandové za korektury a Nikolasovi, rodině a kolegům za podporu.

Obsah

1 Úvod **3**

2 Dostupný software **4**

2.1 Požadavky na integraci nástrojů pro automatizaci testování mobilních aplikací 4
2.2 Nástroje pro psaní automatizovaných testů uživatelského prostředí mobilních aplikací . . . 6
2.3 Frameworky pro zápis scénářů v jazyce Gherkin . . . 8
2.4 Nástroje pro automatizaci řízení automatických testů . . . 9
2.5 Nástroje pro průběžnou integraci . . . 10
2.6 Systémy pro kontrolu verzí . . . 12

3 Vybraný software **13**

3.1 Appium . . . 13
3.1.1 Architektura nástroje Appium . . . 13
3.2 Jenkins . . . 14
3.2.1 Základní vlastnosti Jenkins . . . 14
3.2.2 Konfigurace a spouštění buildů . . . 15
3.2.3 Reporting . . . 15
3.3 Serenity-BDD . . . 15
3.3.1 Reporty . . . 16
3.4 jBehave . . . 17
3.4.1 Dostupné syntaxe pro zápis příběhů a scénářů . . . 17
3.4.2 Vázání kroků na metody Javy . . . 17
3.5 Bitbucket . . . 18
3.5.1 Webhook . . . 18
3.6 Apache Maven . . . 18
3.7 Prostředí . . . 19

4 Návrh řešení **20**

4.1 Průběh integrace . . . 20
4.1.1 Vývojář a nástroj Bitbucket . . . 20
4.1.2 Bitbucket a integrační server s nástrojem Jenkins . . . 20
4.1.3 Jenkins . . . 20
4.1.4 Jenkins a Serenity . . . 21
4.1.5 Serenity a jBehave . . . 21
4.1.6 jBehave a metody Java objektu . . . 21
4.1.7 Appium klient a Appium server . . . 21
4.1.8 Appium server a testovací zařízení . . . 23

4.1.9 Jenkins a vývojář . . . 23

5 Implementace integrovaného systému **24**
5.1 Implementace klienta pro Appium server . . . 24
5.1.1 Vytvoření Maven projektu . . . 24
5.1.2 Struktura projektu . . . 27
5.1.3 Testovací třídy . . . 27
5.1.4 Kroky jBehave . . . 27
5.1.5 PageObject . . . 28
5.1.6 Nalezení identifikátorů elementů . . . 29
5.1.7 Spuštění projektu . . . 29
5.2 Konfigurace prostředí . . . 29
5.3 Konfigurace vzdáleného repozitáže ve službě Bitbucket . . . 31
5.4 Instalace Appium serveru . . . 31
5.5 Instalace a konfigurace nástroje Jenkins . . . 32
5.5.1 Konfigurace serveru Jenkins . . . 32
5.5.2 Vytvoření Jenkins projektu pro build mobilní aplikace . . . 33
5.5.3 Vytvoření Jenkins projektu pro spuštění testů . . . 34

6 Ověření funkčnosti systému **35**
6.1 Příkladové aplikace . . . 35
6.1.1 Průběh ověření . . . 35
6.1.2 Výsledek ověření . . . 35
6.2 Reálná aplikace . . . 36
6.2.1 Průběh ověření . . . 36
6.2.2 Výsledek ověření . . . 36
6.3 Reálné nasazení . . . 36

7 Závěr **37**

Literatura **38**

A Spuštění integrovaného systému **41**

Kapitola 1

Úvod

Automatizace je v dnešní době jedním z hojně diskutovaných témat v rámci softwarového vývoje. Spousta velkých společností věnuje úsilí a čas do automatizace – především do automatizace testování[21]. Pomalu se do podvědomí dostávají i principy automatizace průběžné integrace, které pomáhají automatizovat dodání produktu zákazníkům s minimalizací času a lidských zdrojů potřebných pro spolehlivé zkompilování, otestování a nasazení vyvíjených aplikací do produkčního prostředí[22]. Některé společnosti dnes již experimentují i s automatizací samotného psaní zdrojového kódu, ale to je prozatím vize budoucnosti[6].

Jak již bylo řečeno, hlavní motivací automatizace je úspora času a lidských zdrojů, a tedy zmenšení nákladů na vývoj. Údržba automatizovaných testů je méně nákladná a klade menší nároky na testery než manuální, zdlouhavé a mnohdy stereotypní testování[33]. V ušetřeném čase se může tester věnovat zdokonalování testů, jejich scénářů a příběhů, nebo samotných nástrojů pro automatizaci. Podobně tak automatizace dodání šetří drahý čas vývojářů-programátorů, kteří by jinak museli svůj čas investovat do přípravy, kompilování a správy výsledných produktů-aplikací. Díky zájmu velkých firem o toto odvětví vývoje vzniklo množství nástrojů pro automatizaci testování a průběžné integrace[8]. Je tedy z čeho vybírat a je nutné mít jasné požadavky a kritéria pro výběr a integraci těchto nástrojů.

Je tak na místě využít výhod obou automatizací a integrovat je v jeden funkční celek. Díky tomu můžeme mít maximálně automatizované řešení pro vývojářské týmy[32].

Tato bakalářská práce se věnuje automatizaci testování uživatelského prostředí mobilní aplikace a automatizaci dodání produktu zákazníkům a to především jejich propojení. Takovým propojením vznikne integrovaný systém, který usnadní práci celému týmu vývojářů podílejících se na vývoji mobilní aplikace.

Práce je rozdělena do sedmi kapitol. V následující kapitole 2 jsou specifikovány požadavky na vytvářený integrovaný systém a přehled dostupného software s krátkým popisem jednotlivých kategorií nástrojů a jejich zástupců.

V kapitole 3 jsou následně vybrány nástroje vhodné pro integraci do systému a detailněji popsány jejich možnosti.

Kapitola 5 popisuje samotnou implementaci a konfiguraci integrovaného systému, která je následně ověřena v kapitole 6.

Kapitola 2

Dostupný software

V této části práce budeme diskutovat nad dostupnými nástroji. K dispozici je nepřeberné množství různých nástrojů od různých vývojářských týmů, které většinou nástroje vyvíjely pro lokální potřeby a později je zveřejnily buď jako komerční služby, nebo jako open-source projekty nadále vyvíjené komunitou utvořenou kolem nich[8]. Nelze obsáhnout zcela všechny dostupné nástroje, budu se tedy zaměřovat primárně na ty stále udržované a vyvíjené a především používané nástroje. Pro každý nástroj uvedu jeho klady a zápory a krátký popis, v čem se nástroj liší od ostatních. Nástroje jsou rozděleny do podkapitol, každá z nich obsahuje nástroje stejného zaměření.

Zdroje, ze kterých jsem čerpal při vytváření následujícího přehledu dostupného software, jsou uvedeny vždy na konci kapitoly věnované jednotlivému nástroji.

Před samotným přehledem dostupných nástrojů je ovšem nutné specifikovat si požadavky, které definují integrovaný systém, který má být implementován.

2.1 Požadavky na integraci nástrojů pro automatizaci testování mobilních aplikací

V této kapitole se budeme věnovat požadavkům na integraci nástrojů. Tyto požadavky nám následně poslouží jako kritéria při výběru vhodných nástrojů a pro návrh jejich integrace.

1. Chceme testovat pouze aplikace, které mají interaktivní grafické uživatelské rozhraní (anglicky *Graphical User Interface*, v práci budu tedy nadále používat zkratku UI) a jsou dostupné z oficiálních distribučních služeb, jako je Google Play pro platformu Android či App Store pro platformu iOS, případně aplikace vyvíjené lokálně s přístupem ke zkompilovaným souborům *APK (Android Application Package)* pro Android nebo *IPA (iPhone Application Archive)* pro iOS.

2. Testy musí být implementovatelné v jazyce, ve kterém je vyvíjena samotná aplikace, pokud existuje vhodné a efektivní řešení. Motivací je eliminace nutnosti vývojářů a testerů používat další programovací jazyk.

   (a) **Java** je nejčastějším programovacím jazykem používaným při vývoji aplikací pro platformu Android[9].

   (b) **Objective C** je jeden z nejrozšířenějších programovacích jazyků na platformě iOS[2].

   (c) **Swift** je programovací jazyk vyvinutý přímo pro platformu iOS, případně další platformy firmy Apple[15].
   (d) **C# a C++** jsou využívány především při vývoji aplikací pro platformy Windows Phone, případně při vývoji aplikací pro univerzální platformu Windows *(Universal Windows Platform, zkráceně UWP)*.

3. Nástroj pro psaní automatizovaných testů musí umožňovat multiplatformní testování. Android a iOS jsou základní platformy, které musí nástroj podporovat.

4. Příběhy *(z anglického výrazu stories, formátovaný slovní popis funkcionality)* chováním řízeného vývoje (v práci dále budu používat zkratku BDD z anglického *behavior-driven development*) musí být srozumitelné bez nutnosti hlubší znalosti kódu a programovacího jazyka, ve kterém jsou samotné testy implementované.

5. Pro automatizaci bude možné použít jak reálné mobilní zařízení, tak i emulátory nebo simulátory.

6. Výstupy testů musí být shromažďovány do přehledného reportu. Report musí obsahovat povinné informace o výsledku jednotlivých testů:
   (a) výsledky jednotlivých příběhů,
   (b) čas potřebný k dokončení testů,
   (c) konfigurace zařízení, na kterém testy proběhly:
      i. platforma,
      ii. model mobilního zařízení,
      iii. typ (emulátor, simulátor nebo reálné zařízení),
      iv. stav zařízení po skončení testu:
         A. hodnota baterie,
         B. typ síťového připojení (Wifi nebo datové),
   (d) snímky obrazovky testovacího zařízení po každém kroku,
   (e) údaje o stroji, na kterém testy byly spouštěny:
      i. verze operačního systému,
      ii. verze knihoven aplikovaných při spuštění testů.

7. V případě vyvíjení mobilní aplikace lokálně musí být její zdrojové kódy udržovány pomocí verzovacího systému. V hlavní větvi repozitáře se bude udržovat funkční zdrojový kód, který po zkompilování tvoří spustitelnou a funkční aplikaci, a v kterékoliv fázi vývoje bude možné zkompilovanou aplikaci z hlavní větve dodat zákazníkům.

8. Testy musí být možné spustit při každém sloučení vývojářské větve do větve hlavní.

9. Integrační server s veškerými nástroji musí být provozován lokálně.

10. Řešení musí být vhodné pro použití v agilním prostředí, které je specifické krátkými iteračními cykly a zapojením zákazníka do vývoje[30].

11. Integraci nástrojů a automatizované testy musí být možné nasadit do již existujícího reálného prostředí.

2.2 Nástroje pro psaní automatizovaných testů uživatelského prostředí mobilních aplikací

Nástrojů pro automatizaci testů uživatelského prostředí mobilních aplikací existují spousty. Jsou například dostupné cloudové služby zajišťující spouštění testů vzdáleně na cloudových farmách mobilních zařízení. Tyto nástroje jsou většinou placené, ale často jsou ceny těchto služeb nižší než náklady na vlastní mobilní farmu. Výhodou také je, že tyto služby mají většinou široký repozitář mobilních zařízení, a dokážou tedy zajistit testování na různém hardwaru. Hlavními zástupci této kategorie jsou *Sauce Labs*, *Perfecto*, *Xamarin Test Cloud* a *AWS Device Farm*. Všechny tyto společnosti nabízejí podobné služby – testování na emulátorech, simulátorech a reálných zařízeních v jimi spravovaných mobilních farmách, vlastní konfiguraci použitých nástrojů a frameworků či analýzu výstupů testů. Tyto služby většinou nabízejí kompletní hotové řešení nad samotnými automatizačními nástroji a frameworky pro automatizaci, kterými se budeme dále zabývat v této práci.

Mezi nejznámější a nejpoužívanější frameworky patří *Appium*, který vychází z velmi rozšířeného frameworku *Selenium*, podporujícího automatizaci testování webových stránek. Mezi další frameworky pro mobilní automatizaci patří například *Calabash*, který je vyvíjen firmou Xamarin. Dalšími zástupci jsou *Espresso*, *Robotium*, *Selendroid*, *iOS-driver*, *XCTest*, *Monkeytalk*. Rozdíly mezi frameworky jsou většinou v podporovaných platformách, univerzálnosti, programovacím jazyce, ve kterém je implementován, potřebou zdrojových kódů při kompilaci testů, míry abstrakce při psaní testů a podobně.

Appium

Appium je jedním z nejpoužívanějších řešení automatizace testování mobilních aplikací. Neomezuje jazyk pro psaní testů. Pro interakci s aplikacemi používá WebDriver API, které je součástí nástroje Selenium, a JSON Wire Protocol. Jednotlivé platformy podporuje pomocí různých ovladačů – driverů, jako například *UIAutomator* pro Android nebo *XCUITests* pro iOS. [20][36]

+ volitelnost jazyka pro tvorbu testovacích skriptů
+ podpora platforem Android, iOS a Windows
+ aktivní komunita kolem projektu
− konfigurace Appium serveru při spouštění testů zpomaluje jejich provedení

Calabash

Calabash je dalším komplexním nástrojem s jednoduchými provozními nároky. Calabash integruje BDD framework Cucumber (viz kapitola 2.3), čímž je psaní testů nenáročné, s minimálními nároky na znalost programovacích jazyků. Umožňuje testovat jak nativní, tak hybridní aplikace, k čemuž používá Robotium (viz dále) pro testy pro platformu Android, nebo Frank (viz dále) pro platformu iOS. Nástroj je udržován firmou *Xamarin*. [42]

+ podpora platforem Android i iOS
+ integrovaný BDD framework Cucumber
+ testy lze implementovat v jazyce Java nebo Ruby
− komunita kolem projektu není příliš aktivní

Espresso

Espresso je nástroj mající za cíl zjednodušit tvorbu UI testů, udržovaný firmou Google, která stojí za vývojem samotného Androidu. Espresso API je jednoduché a vychází z frameworku *Android Instrumentation*. [1]

- \+ jednoduché API
- \+ možnost nahrávání kroků testů
- \+ rychlost testů – nepotřebuje čekání a uspání
- \+ integrovaný v Android Studio
- − pouze pro platformu Android

Robotium

Robotium býval jedním z nejrozšířenějších nástrojů pro automatizaci testů na platformě Android, ale dnes již vývojáři dávají přednost jiným, udržovanějším a komplexnějším řešením. Robotium rozšiřuje JUnit testy o nové metody pro UI testování. API vychází z nástroje pro webovou automatizaci Selenium. Umožňuje testovat nativní, hybridní i webové aplikace. Pro interakci používá JSON Wire Protocol. [43]

- \+ podobný nástroji Selenium
- \+ umožňuje testovat aplikace bez přístupu ke zdrojovým kódům
- − pouze pro Android

Selendroid

Selendroid, jak už název napovídá, také vychází z nástroje Selenium. Jeho hlavní předností jsou možnosti propojení s původním Seleniem a použití stejného API, díky čemuž je pro vývojáře zkušeného v testování v Seleniu zavedení Selendroid do týmového procesu jednoduché. Pro interakci používá JSON Wire Protocol. [35]

- \+ možnost integrace se Selenium Grid - testy lze spouštět paralelně na více zařízeních
- \+ testy jsou psány pomocí Selenium 2 API
- − nepodporuje nejnovější Android API, oficiální dokumentace uvádí poslední podporované Android API 19
- − pouze pro Android

iOS-driver

iOS-driver podobně jako Selendroid používá pro automatizaci Selenium API, navíc s podporou WebDriver API. Umožňuje testovat nativní, hybridní i webové mobilní aplikace. Pro interakci používá JSON Wire Protocol. [23]

- \+ integrace se Selenium Grid
- \+ pro testy lze použít reálné zařízení či simulátor
- − pouze pro iOS

XCTest

Nástroj XCTest je přímo vyvíjen vývojáři firmy Apple. Je součástí vývojářského balíku XCode. Umožňuje vytvářet jednotkové, performační a UI testy. [3]

+ komplexní testovací framework udržovaný vývojáři Apple
- nutnost spouštění testů na systémech firmy Apple – Mac OSX
- pouze pro iOS

Frank

Frank je propagován jako „Selenium pro iOS". Podobně jako Calabash integruje BDD nástroj Cucumber, díky kterému je psaní testů jednoduché. [14]

+ podporuje snadnou integraci s nástroji pro průběžnou integraci
- potřeba zdrojového kódu pro integraci frameworku
- malá aktivita kolem projektu
- pouze pro iOS

EarlGrey

EarlGrey je obdoba nástroje Espresso pro platformu iOS. Je také vyvíjen firmou Google, který tento nástroj původně vyvinul pro potřeby testování vlastních produktů. [17]

+ rychlé testy bez nutnosti čekání či uspání podobně jako u nástroje Espresso
- pouze pro iOS

2.3 Frameworky pro zápis scénářů v jazyce Gherkin

Samotné testy psané běžným programovacím jazykem mohou být nepřehledné pro člověka, který se nevěnuje programování[31]. Proto existují frameworky pro podporu *chováním řízeného vývoje*, které umožňují psát testy v jazyce více se podobajícím běžné lidské řeči. BDD přístup má za cíl zvýšit kvalitu vyvíjeného software a snížit náklady na údržbu a umožnit i netechnickým exekutivním pozicím týmu podílet se na vytváření akceptačních testů. Pro tyto účely vznikl jazyk *Gherkin*[37]. Syntaxe jazyka Gherkin je řádkově orientována, podobně jako u jazyka Python. Pokud není řádek komentářem, první slovo řádku je považováno za klíčové slovo následované zbytkem řetězce a znakem konce řádku. Gherkin je byznysově čitelný, doménově specifický jazyk (z anglického *Business Readable, Domain Specific Language*)[13]. Dostupné frameworky pro integraci jazyka Gherkin a automatizovaných testů jsou například *jBehave*, *Cucumber*, *Behat* a další. Tyto frameworky vážou kroky zapsané v jazyce Gherkin na metody jazyka použitého pro psaní testů. Příklad implementace vázání kroků na metody testovacích objetků je uveden v kapitole jBehave 3.4.

```
Given application is opened on device
When the user starts activation
Then the activation prompt pops up
```

Příklad 2.1: Zápis v jazyce Gherkin

jBehave

Framework jBehave je kompletně implementovaný v jazyce Java. Umožňuje zapisovat příběhy v jeho vlastní syntaxi jBehave nebo v syntaxi Gherkin. Automaticky vytváří přehledné výstupy testů ve formátech HTML, TXT nebo XML. V porovnání s ostatními relevantními nástroji má obsáhlou dokumentaci. [24]

+ implementovaný v jazyce Java
+ generování výstupů testů

Cucumber

Z minulých kapitol lze usoudit, že Cucumber je oblíbený pro integraci s nástroji pro automatizaci. Cucumber je obecně považovaný za jednodušší nástroj než jBehave. Oproti němu se od začátku více drží původní specifikaci syntaxe Gherkin, i když dnes už oba nástroje dorovnali své nedokonalosti. Cucumber je původně implementovaný v jazyce Ruby, ale dnes je již dostupná i verze kompletně implementovaná v jazyce Java – Cucumber-JVM. [11]

+ generování výstupů testů
– původně implementovaný kompletně v jazyce Ruby (dnes je již dostupná oficiální implementace v jazyce Java)

Behat

Behat je implementován v jazyce PHP. Podobně jako předchozí dva nástroje podporuje syntaxi Gherkin. Vývoj Behat je stále aktivní, přičemž se jedná o nejpoužívanější nástroj v aplikacích implementovaných v PHP. [28]

+ aktivní vývoj
– implementovaný v jazyce PHP

SpecFlow

SpecFlow je nástroj primárně vyvíjený pro .NET projekty. Obdobně jako všechny nástroje této kategorie umožňuje zápis syntaxí Gherkin. V rozšířené verzi SpecFlow+ nabízí formátované výstupy testů. [40]

+ používá oficiální Gherkin parser z nástroje Cucumber
– implementovaný v jazyce .NET
– placené funkce

2.4 Nástroje pro automatizaci řízení automatických testů

S růstem konfigurací a kombinací frameworků pro psaní automatizovaných testů vznikla knihovna, která pomáhá se správou těchto frameworků a částečně jejich funkce automatizuje. *Serenity* usnadňuje aplikaci BDD principů do automatizovaných akceptačních testů.

Serenity

Nástroj Thucydides, později přejmenovaný na Serenity kvůli jeho výslovnosti[10], má za cíl zjednodušit psaní a údržbu automatizovaných akceptačních a regresních testů. Kromě jeho vlastností, jako je například správa stavu testu mezi jednotlivými kroky, integracemi s dalšími nástroji a možnosti paralelního spouštění testů, je pro nás nejdůležitější jeho generování pokročilých reportů, které jsou oproti výstupům nástrojů jako jBehave, Cucumber a podobně mnohem detailnější a přehlednější. [38]

+ generování kvalitních, detailních a přehledných reportů
+ integrace s jBehave
+ podpora Selenium/WebDriver
+ možnost implementace v jazyce Java
+ aktivní komunita a vývoj projektu

2.5 Nástroje pro průběžnou integraci

Manuální kompilace aplikace, spouštění a vyhodnocení testů jednotkových, integračních či UI není vhodné v agilním prostředí, ve kterém se takové úkony opakují s celkem velkou frekvencí.[30] Lepším řešením je využití nástroje pro průběžnou integraci. Takový nástroj řídí všechny zmíněné operace automaticky, a zrychluje tak celý vývoj. Na základě konfigurace je schopný připravit zdrojové kódy z repozitáře verzovacího systému, zkompilovat aplikaci a spustit nad ní automatizované testy. Mezi nejpoužívanější nástroj pro průběžnou integraci patří *Jenkins*.[26] Dalšími hojně používanými zástupci jsou například *Travis*, *TeamCity* či *GitLab CI*.

Jenkins

Jenkins jako jeden z prvních nástrojů pro průběžnou integraci vzešel z projektu Hudson a postupně ho nahradil.[12] Je udržován jako open-source. Většina logiky je řešena pomocí zásuvných modulů z rozsáhlé knihovny, díky čemuž je Jenkins flexibilní a ve velké míře přizpůsobitelný vlastním požadavkům. Je implementován v jazyce Java. [26]

+ velká knihovna s více než tisícem zásuvných modulů
+ v základu obsahuje pouze nejnutnější moduly
– závislost na zásuvných modulech dodávaných třetí stranou
– horší replikovatelnost na případné další instance Jenkins

Travis CI

Travis CI slouží k zajištění rychlé odezvy o kvalitě a úspěšnosti změn v repozitářích vytvořených vývojářem. Díky tomu vývojář může rychleji reagovat na případné chyby a nalezené problémy, čímž urychluje celý proces vývoje. [41]

+ rychlá odezva pro vývojáře
– placená služba pro komerční projekty

TeamCity

Nástroj TeamCity vyvíjený týmem JetBrains má za cíl urychlení a zdokonalení cyklu pro dodání softwaru. Vývojář může průběžně kontrolovat stav testů, pokrytí kódu testy, statistiky buildů[1], jejich dobu trvání a podobně. [27]

+ licence zdarma s omezeným počtem konfigurací buildů
+ možnost paralelního spouštění jednotlivých úkolů
– placené další konfigurace

GitLab CI

GitLab CI je součástí komplexní služby GitLab, která nabízí verzovací systém zdrojových kódů. Konfigurace buildů je udržována přímo v repozitáři se zdrojovými kódy. Samotné spouštění buildů je zprostředkováno takzvanými *runners*, které mohou běžet nezávisle na sobě na více strojích. Díky tomu jsou snadněji replikovatelné na jiných strojích, kde stačí mít k dispozici spuštěný runner a přístup k repozitáři obsahujícím konfiguraci buildu v YAML syntaxi. [34]

+ replikovatelnost
– použitelný pouze s GitLab produkty

Bamboo

Bamboo podobně jako GitLab CI je spojen s produkty firmy vyvíjející systém propojených nástrojů pro podporu softwarového vývoje. V případě Bamboo je to firma Atlassian. Pro spouštění buildů používá Bamboo Agents. Tito agenti mohou existovat lokálně jako součást Bamboo serveru, nebo vzdáleně na jiných strojích. Agenti jsou přidruženi k různým konfiguracím buildů. Díky tomu je možné využít různá prostřední pro různé buildy. Například prostředí Mac OSX pro build aplikace pro platformu iOS a prostředí Windows pro build aplikace pro Windows. Jde o placenou službu, kde cena závisí na počtu použitých agentů. [5]

+ díky agentům lze využít různá prostředí pro různé buildy
– pouze pro produkty od Atlassian
– placená služba

Bitbucket Pipelines

Obdobnou službu, jakou je Bamboo, nabízí firma Atlassian v podobě *Bitbucket Pipelines*. Jde o integrovaný CI nástroj pro cloudové služby Bitbucket. Oproti Bamboo není nutné nastavovat CI server. Je možné spouštět buildy paralelně, bez nutnosti čekat na volného agenta. Buildy se definují v souboru `bitbucket-pipelines.yml` ve formátu *YAML* uloženém v kořenovém adresáři repozitáře. [4]

+ paralelní spouštění buildů bez nutnosti čekání na volného agenta

[1] Proces, během kterého dochází ke kompilaci aplikace spolu s kompilací testů, spuštění testů, jejich vyhodnocení, vytváření balíčků ze zkompilovaných zdrojových kódů atd.

– pouze pro repozitáře uložené ve službě Bitbucket

– placená služba (zdarma pouze s limitovaným buildovacím časem)

2.6 Systémy pro kontrolu verzí

V dnešní době je běžné používat pro správu verzí zdrojových kódů některý z verzovacích systémů. Existují jejich tři základní typy. Jsou to systémy centralizované, lokální nebo distribuované. Centralizované systémy uchovávají veškerou historii repozitáře na jednom místě – serveru. Lokální systémy existují pouze v jednom souborovém systému. Všichni přispěvatelé tak musí pracovat v tomto jednom daném systému. Naopak distribuované systémy distribuují celou historii mezi všechny stroje, na kterých je repozitář naklonován. Všichni tak mají celou historii lokálně u sebe k dispozici. Git patří mezi nejrozšířenější distribuované verzovací systémy. Na něm založených existuje spousta webových služeb jako například *Bitbucket*, *GitHub*, *CodePlex*. Rozdíly mezi jednotlivými službami nejsou příliš velké, vetšinou pouze v nabízených cenových plánech (z anglického *pricing plans*).

GitHub

GitHub je jeden z nejpoužívanějších služeb pro vzdálené ukládání git repozitářů. GitHub poskytuje zdarma veřejné repozitáře, které jsou hojně využívány open-source projekty. Umožňuje import i repozitářů jiných systémů pro správu verzí, a to SVN, Mercurical nebo TFS. Nabízí například nástroje pro snadnější kolaboraci mezi přispěvateli do repozitáře, možnost vytvořit WIKI k projektu nebo *issue tracker*.

+ spousta nástrojů usnadňující práci s repozitářem

– maximální velikost veřejného repozitáře je 1 GB

– placené privátní repozitáře

GitLab

GitLab je podobná služba jako GitHub. Komunitní verze GitLab Community je poskytována zdarma s určitým omezením. Podobně jako GitHub je i GitLab oblíbený mezi vývojáři open-source projektů. Podporuje import git repozitářů. [16]

+ zdarma veřejné repozitáře

– placené privátní repozitáře

Bitbucket

Podobně jako předchozí dvě zmíněné služby poskytuje Bitbucket úložný prostor a správu pro repozitáře systému pro správu verzí. Bitbucket je jedním z produktů firmy Atlassian a je integrovaný se službou Jira, nabízí integrovaný nástroj pro průběžnou integraci Bamboo a další výhody v rámci skupiny produktů Atlassian. Oproti GitHub a GitLab nabízí Bitbucket zdarma privátní repozitáře pro týmy pěti a méně vývojářů. Přístup dalších vývojářů k repozitářům je zpoplatněný.

+ zdarma privátní repozitáře

+ levnější řešení než GitHub

Kapitola 3

Vybraný software

V této kapitole vybereme nástroje na základě poznatků získaných v předchozích kapitolách. Jako kritéria pro výběr nástrojů poslouží požadavky specifikované v kapitole 2.1.

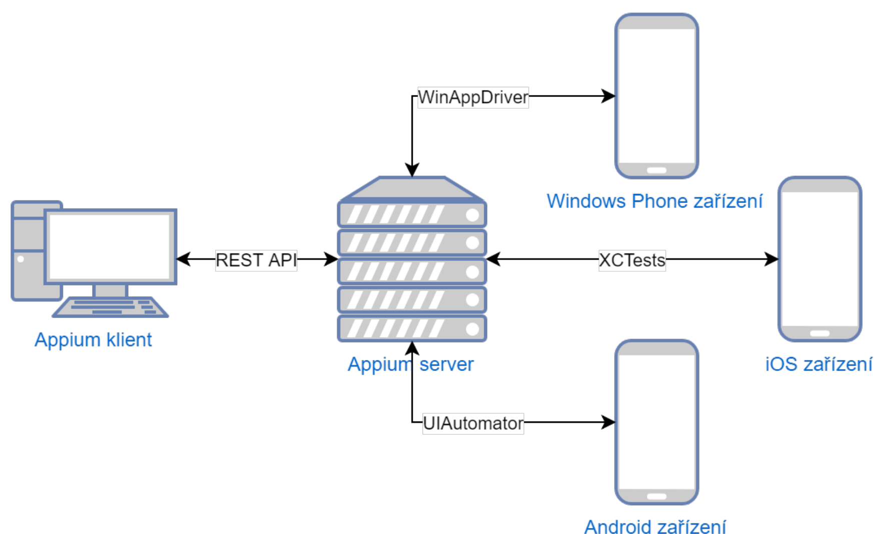
3.1 Appium

Následující kapitola převážně vychází z poznatků získaných z knihy Appium Essentials [20]. Appium je nástroj pro automatizaci aplikací na dvou největších mobilních platformách – Android a iOS. Umožňuje automatizaci nativních aplikací, mobilních webových aplikací spouštěných v mobilním prohlížeči nebo aplikací hybridních, které kombinují nativní funkce s *webview* – možností mobilních platforem zobrazit webový obsah uvnitř nativní aplikace. Appium pro spuštění testů nepožaduje opětovnou kompilaci aplikace ani zásah do zdrojových kódů, a lze tedy automatizovat již zkompilovanou, případně i nainstalovanou aplikaci v mobilním zařízení (viz požadavek 1). Toho je dosaženo použitím automatizačních frameworků poskytovaných přímo vývojáři mobilních platforem. Například pro platformu Android Appium používá *UIAutomator* a *Instrumentation Framework* od Google. Pro iOS to jsou *UIAutomation* a *XCUITest* poskytované firmou Apple. Appium poskytuje jednotné API pro všechny podporované platformy, čímž vyřazuje nutnost znalosti jednotlivých frameworků a jazyků, ve kterých jsou původní frameworky napsané. Toto API vychází z *WebDriver API* vytvořené vývojáři *Selenium*.

3.1.1 Architektura nástroje Appium

Koncept Appium je založen na architektuře server–klient. Server představuje samotné Appium, které pomocí REST API komunikuje s klienty. Klient může být implementován v libovolném jazyce, který umožňuje implementaci REST komunikace. Tím je zároveň umožněno provozovat server vzdáleně. Pomocí Appium lze spouštět testy paralelně na více zařízeních najednou. Tím je možné dosáhnout rychlejších dokončení testů na větším množství zařízení. Architektura Appium je znázorněna diagramem architektury nástroje Appium – obrázek 3.1.

Appium server je implementován v Node.js jako konzolová aplikace. Server naslouchá příkazům od klientů, které následně spouští na připojených testovacích mobilních zařízeních, respektive emulátorech či simulátorech. Výsledný stav po provedení příkazů na cílovém zařízení odesílá zpět klientovi, který je vyhodnotí.



Obrázek 3.1: Architektura nástroje Appium

Appium klient může být, jak již bylo řečeno, implementován v jakémkoliv jazyce umožňujícím REST komunikaci. Appium nabízí podpůrné knihovny pro nejčastěji používané jazyky – Java, Python, Ruby, PHP, JavaScript, C# a Objective C. Tyto knihovny poskytují rozšířený WebDriver protokol.

Appium Desktop je Appium server rozšířený o grafické uživatelské prostředí pro jednodušší ovládání. Jeho instalace není vyžadována a zpravidla neobsahuje aktuální verzi Appium serveru, poněvadž většina úsilí přispěvatelů do open-source projektu je věnována samotnému Appium serveru.

3.2 Jenkins

Jenkins (respektive jeho předchůdce Hudson) je jeden z prvních nástrojů své kategorie a nejrozšířenějším nástrojem pro průběžnou integraci. Jeho součástí je rozsáhlý repozitář zásuvných modulů, díky kterým ho lze maximálně přizpůsobit pro vlastní potřebu. V základní verzi obsahuje pouze nejnutnější moduly pro samotnou administraci Jenkins. Je multiplatformní, neomezuje nás tedy při výběru serveru a je možné ho provozovat na stejném stroji, na kterém zároveň budou probíhat kompilace zdrojových kódů. Podporuje také distribuci kompilačních buildů aplikací – možnost spouštět kompilaci na různých strojích s různými operačními systémy. Toho lze využít při automatizaci různých mobilních platforem, kdy například platforma iOS vyvíjená firmou Apple pro spuštění kompilace požaduje operační systém OS X a naopak platforma Windows Phone požaduje spuštění na systému Windows. Jenkins je implementován v jazyce Java.

3.2.1 Základní vlastnosti Jenkins

Přestože Jenkins nabízí téměř neomezenou flexibilitu přizpůsobení díky více než tisícovce zásuvných modulů v jeho knihovně, mezi jeho stěžejní vlastnosti patří následujících pět bodů.

Integrace se systémy správy verzí Jenkins lze propojit s on-line nástroji VCS, čímž vcelku jednoduše zajistíme, že má vždy aktuální zdrojové kódy, se kterými může dále pracovat. Stačí vygenerovat přístup k repozitáři a o zbytek se postarají zásuvné moduly.

Spouštění buildů Spouštění buildů může být zahájeno buď manuálně, nebo automaticky na základě splnění předem specifikovaných podmínek. Může to být například periodicky, na základě změny stavu jiného projektu, pollingem jiných služeb a podobně.

Spuštění testů Obdobně jako spouštění buildů lze také spouštět testy. Ty se většinou budou spouštět buď periodicky (regresní testy), nebo na základě úspěšnosti buildu některého z přidružených projektů (viz požadavek 8).

Archivace artefaktů Výsledky testů, zkompilované aplikace nebo třeba logy mohou být archivovány na lokální či vzdálené souborové systémy.

Notifikace O změnách stavu projektů lze notifikovat vývojáře, případně další členy týmů, pomocí mailů, automatickými zprávami do týmových kanálů v chatovacích službách jako například *Slack* a další.

3.2.2 Konfigurace a spouštění buildů

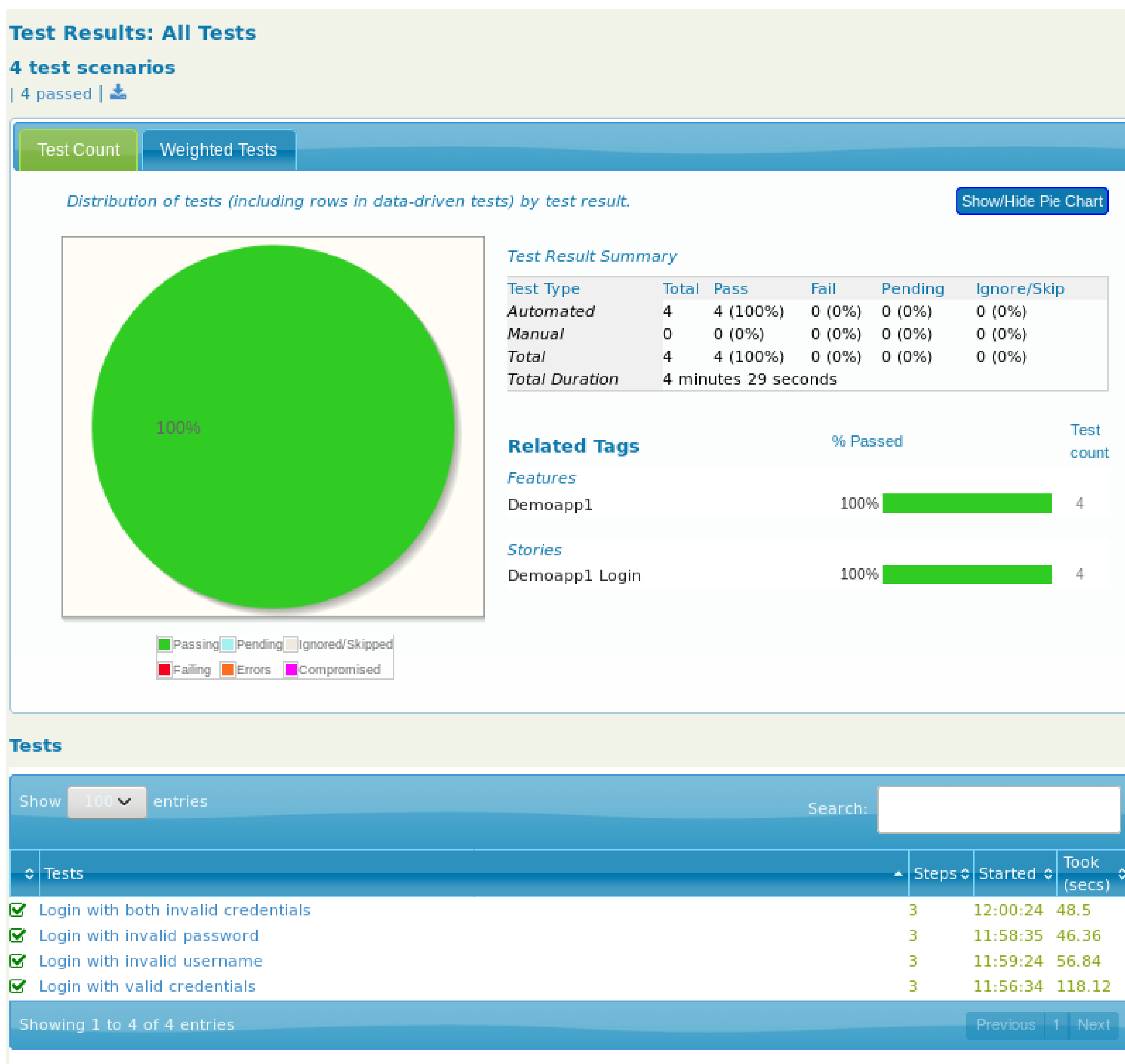
Konfigurace probíhá vytvořením *projektů*, ve kterých lze specifikovat jednotlivé kroky pro kompilaci zdrojových kódů, podmínky pro automatické spouštění buildů a další možnosti zmíněné výše. Jenkins poté spouští build v *exekutorech*. Koncept *slave* buildovacích serverů umožňuje distribuci buildovacích úkolů na různé stroje, čímž je možné řešit vyrovnání vytížení exekutorů, případně potřebu různých prostředí pro různé typy projektů.

3.2.3 Reporting

Jenkins nabízí základní funkce reportingu, které udržují statistiky o stavu buildů – jejich úspěšnost, časy posledních spouštění, trend a podobně. Tyto základní reporty lze rozšiřovat pomocí zásuvných modulů například o trendy výsledků testů, pokrytí kódu jednotkovými testy a podobně.

3.3 Serenity-BDD

Serenity je nástroj pro řízení spouštění testů, jejich integraci s BDD frameworky a generování reportů. Podporuje knihovny Selenium, WebDriver a Appium. Na základě informací z konfiguračního souboru řídí inicializaci WebDriveru a objektů stránek (PageObjects) – návrhový vzor pro podporu zpřehlednění kódu a zjednodušení údržby pomocí oddělení jednotlivých stránek/obrazovek a umožněním definovat očekávané elementy vyskytující se na specifické stránce. Serenity podporuje integraci s nástrojem jBehave, případně JUnit v Javě, což nám usnadňuje integraci následujícího nástroje jBehave.



Obrázek 3.2: Příklad formátovaného reportu nástrojem Serenity

3.3.1 Reporty

Hlavní motivací použití tohoto nástroje je jeho schopnost generování pokročilých, přehledných a zároveň detailních reportů o výsledku testů. (viz požadavek 6) Příklad reportu vygenerovaného nástrojem Serenity lze vidět na obrázku 3.2.

Tyto reporty jsou díky jejich struktuře praktické při hledání problémů s neprocházejícími testy. Testy jsou roztříděny do různých kategorií podle jejich adresářové struktury, podle výsledků, či podle metadat testů. Report je generován jako webová stránka v HTML. Lze je tedy jednoduše odkazovat. Poskytuje také různé úrovně detailu. Lze tedy zobrazit seznam všech spuštěných testů s jejich výsledky aniž bychom museli procházet detail testových kroků a jejich výsledků. Pokud ovšem takový detailní popis potřebujeme, například při řešení neprocházejícího testu, je nám k dispozici detailní náhled se snímky obrazovky při jednotlivých krocích a výpisem chybových hlášení z příkazové řádky.

Výsledky lze také sdílet s některými nástroji pro správu testů, jako například *TestRail*. Tím lze dosáhnout ještě větší míru automatizace.

3.4 jBehave

Nástroj jBehave váže jednotlivé kroky příběhu a jeho scénářů psaných v jazyce Gherkin na metody objektu v Javě. Je tedy vhodný pro integraci s Appium testy, na které lze kroky vázat. (viz požadavek 4)

3.4.1 Dostupné syntaxe pro zápis příběhů a scénářů

jBehave umožňuje zapisovat příběhy a scénáře dvěma způsoby. Buď syntaxí jBehave, nebo nepřímo syntaxí Gherkin. V druhém případě je zápis nejdříve transformován na zápis syntaxí jBehave a až poté parsován a vázán na metody testovacích tříd[25]. To je dáno tím, že původně jBehave syntaxi Gherkin nepodporoval vůbec, načež byla jeho podpora dodatečně implementována.

3.4.2 Vázání kroků na metody Javy

Vázání je provedeno pomocí anotací metod ve třídách Javy, které asociují spustitelnou metodu s kandidátem kroku (z anglického *candidate step*). Každý kandidát kroku může korespondovat vždy jen s jednou metodou a jedním typem kroku. Porovnávání textových řetězců je prováděno pomocí regulárních výrazů.

```
@When("user clicks on button")
public void whenUserClicksOnButton(){
        driver.findElement(By.xpath(xpath)).click();
}
```

Příklad 3.1: Anotovaná metoda Java objektu

Styl reprezentace testů

jBehave předpokládá *Když-Pokud-Potom* (*Given-When-Then*, dále v textu budu používat zkratku GWT) zápis scénáře využitím *specifikace příkladem* (z anglického *specification by example*).

```
A story is a collection of scenarios

Narrative:
In order to communicate effectively to the business some functionality
As a development team
I want to use Behaviour-Driven Development

Lifecycle:
Before:
Given a step that is executed before each scenario
After:
Outcome: ANY
Given a step that is executed after each scenario regardless of outcome

Scenario: A scenario is a collection of executable steps of different type

Given step represents a precondition to an event
When step represents the occurrence of the event
```

```
Then step represents the outcome of the event
```

Příklad 3.2: Syntax jBehave

```
Feature: A story is a collection of scenarios

Narrative:
In order to communicate effectively to the business some functionality
As a development team
I want to use Behaviour-Driven Development

Background:

Given a step that is executed before each scenario

Scenario: A scenario is a collection of executable steps of different type

Given step represents a precondition to an event
When step represents the occurrence of the event
Then step represents the outcome of the event
```

Příklad 3.3: Syntax Gherkin

3.5 Bitbucket

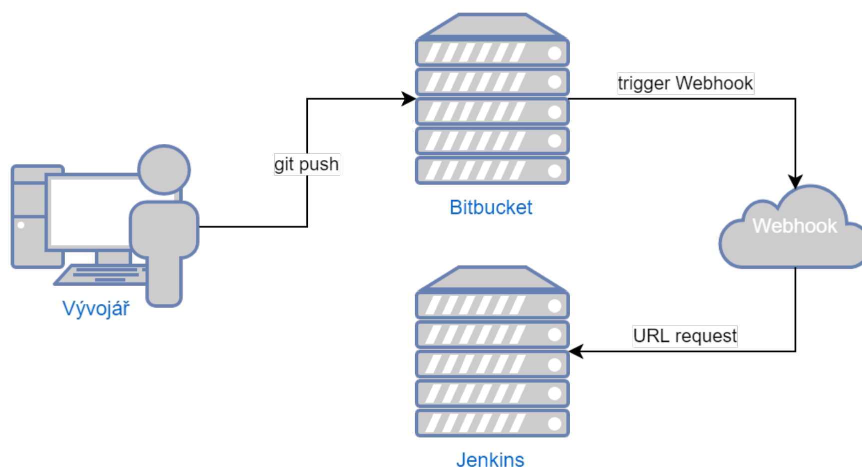
Bitbucket je jedna z nejrozšířenějších služeb provozujících webovou službu pro správu verzí zdrojových kódů založených na systému Git. Rozdíly mezi webovými službami tohoto typu jsou většinou pouze v licenčních podmínkách a v podmínkách užívání. Byl zvolen Bitbucket, protože nabízí propojení s Jenkins pomocí WebHooks, privátní repozitáře a kolaboraci s členy týmu. Funguje zároveň jako záloha zdrojových kódů.

3.5.1 Webhook

Webhook nabízí možnost konfigurovat cloudovou službu Bitbucket k posílání zpráv externím službám, kdykoliv se uskuteční některá z předem definovaných událostí. Jde v podstatě o opak *pollingu*, kdy je nutné v pravidelných intervalech kontaktovat vzdálenou službu a zjišťovat, zda v ní proběhla očekávaná změna. U Webhooks naopak hlídá změnu sama služba, která v případě očekávané události odešle zprávu na předem specifikovanou URL. Tím lze v našem případě dosáhnout toho, že pokud se v repozitáři v Bitbucket projeví změna například v podobě sjednocení větví, je odeslána zpráva nástroji Jenkins, který následně spustí přidružený projekt. (viz požadavek 8)

3.6 Apache Maven

Maven je nástrojem pro správu projektů založených na konceptu *project object model* (POM). Je pro nás užitečný svou možností správy závislostí na externích knihovnách. Jeho nasazení nám usnadní řešení problémů spojených s nekompatibilitou jednotlivých vybraných knihoven a nástrojů. Zároveň nám poslouží ke spouštění testů.



Obrázek 3.3: Bitbucket Webhook

3.7 Prostředí

Na základě minimálních požadavků vybraných nástrojů a knihoven je nutné použít Java SE Development Kit 8 nebo novější. Pro kompilaci aplikací pro platformu Android je potřeba mít k dispozici nástroje z Android SDK a pro iOS stroj se systémem Mac OSX.

Vhodným operačním systémem na základě požadavků nástrojů vybraných výše je linuxová distribuce Debian, která v základním stavu neobsahuje žádné zbytečné programy ani balíčky. Jako jedna z nejdéle existujících linuxových distribucí má velkou komunitní základnu. Většina běžných nástrojů podporuje Debian, takže ho lze přizpůsobit vlastím potřebám[39].

Kapitola 4

Návrh řešení

V této kapitole se budeme zabývat návrhem řešení pomocí vybraných nástrojů z předchozí kapitoly. Navrhneme řešení pro integraci těchto nástrojů, uvedeme si vztahy mezi nimi a průběh integrace.

4.1 Průběh integrace

Následuje návrh vztahů mezi jednotlivými účastníky integrace nástrojů. Jednotlivé vztahy popsané níže korespondují s návrhovým diagramem (obrázek 4.1) a sekvenčním diagramem (obrázek 4.2), který znázorňuje průběh samotné integrace a komunikace mezi jednotlivými nástroji.

4.1.1 Vývojář a nástroj Bitbucket

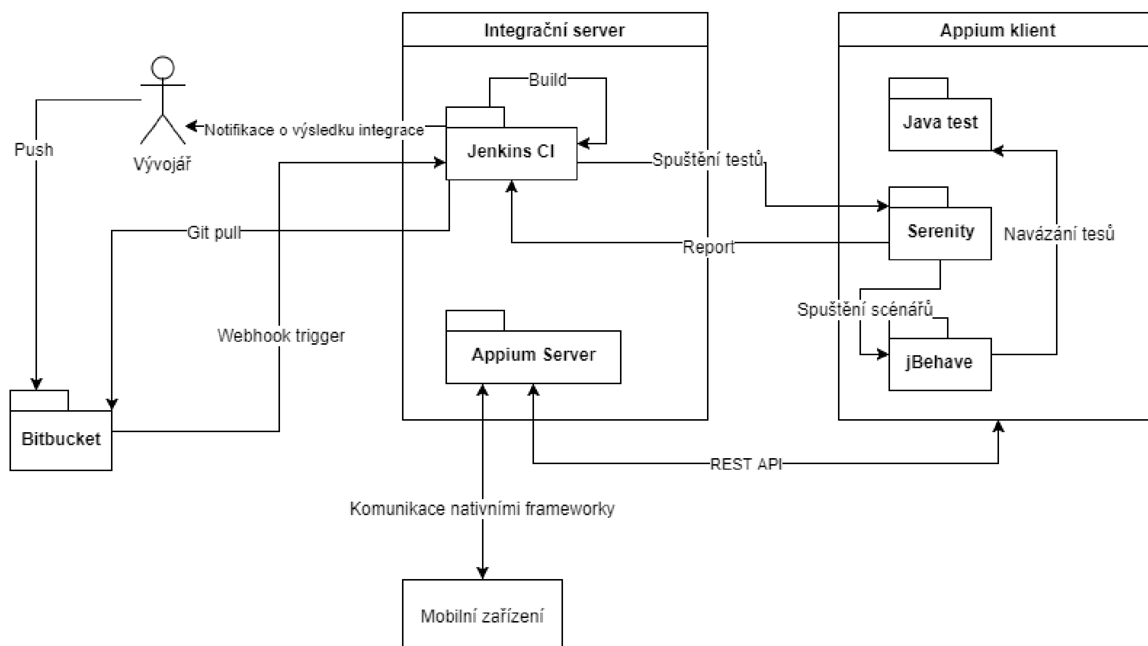
Představme si vývojáře, tedy účastníka našeho problému, vyvíjejícího určitou aplikaci. V agilním prostředí (viz požadavek 10) je běžné nepřetržité dodávání nové funkcionality a neustálý vývoj kódu za použití konceptu extrémního programování, kdy na jedné funkcionalitě pracuje více programátorů zároveň. Je tedy pravděpodobné, ne-li nutné, užití systému pro správu verzí zdrojových kódů. V našem případě tomu je Git a on-line služba Bitbucket (viz nástroj Bitbucket 3.5). Vývojář při každé změně kódu zapíše tuto změnu do dedikované větve repozitáře pro danou funkcionalitu. Po dokončení příběhu, který definuje novou funkcionalitu, vytvoří požadavek na sloučení této dedikované větve do hlavní větve, ve které se udržuje kompletní, funkční aplikace.

4.1.2 Bitbucket a integrační server s nástrojem Jenkins

Po úspěšném sloučení větví se pomocí Bitbucket Webhook služby voláním aktivuje Jenkins CI, který zareaguje na změny v repozitáři stažením nejnovější verze.

4.1.3 Jenkins

Po získání aktuálního repozitáře, spustí Jenkins build z tohoto repozitáře, případně další úkony konfigurované v rámci Jenkins projektu. V případě, že buildování skončí úspěchem, zkompilované aplikace se uloží na integračním serveru v archivu Jenkins. V případě neúspěchu zde integrace končí a vývojář je notifikován o neúspěšné operaci s výpisem konzole o průběhu integrace a spuštěných kroků.



Obrázek 4.1: Návrh řešení - vztahy nástrojů

4.1.4 Jenkins a Serenity

případě úspěšného dokončení buildu pokračuje Jenkins spuštěním testů pomocí zásuvného modulu pro integraci s nástrojem Serenity. Serenity dále řídí průběh testů a na závěr vytvoří report s výsledky testů, který předá zpět nástroji Jenkins, který ho archivuje a zprostředkuje vývojáři.

4.1.5 Serenity a jBehave

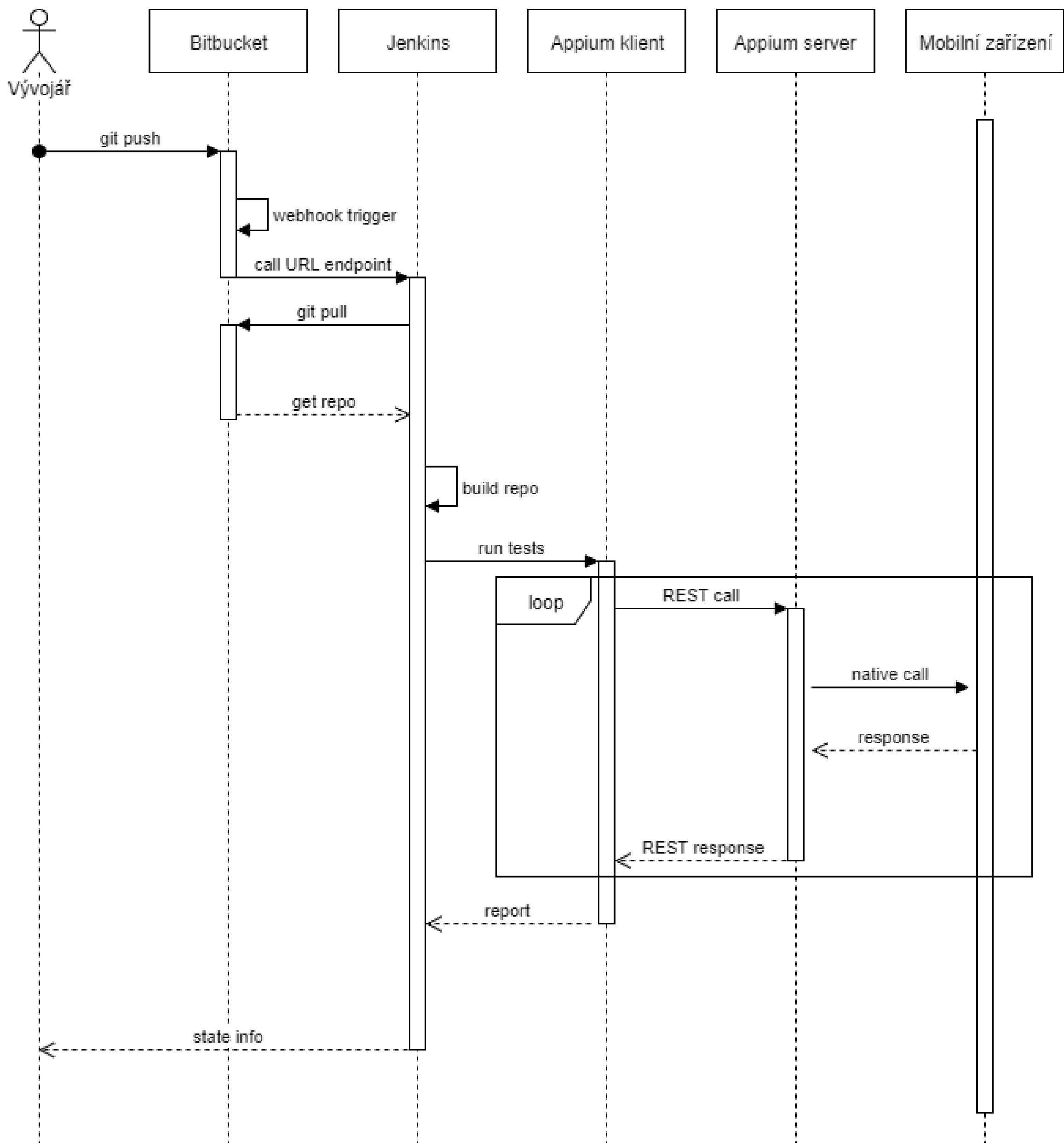
Jak již bylo řečeno, jednou z vlastností Serenity je řízení inicializace testů. Spustí tedy pomocí jBehave příběhy definované v repozitáři.

4.1.6 jBehave a metody Java objektu

jBehave naváže metody definované v Java objektu pomocí principů popsaných v kapitole jBehave 3.4, testy vyhodnotí a výsledky pošle zpět Serenity, který je následně zpracuje do přehledných reportů.

4.1.7 Appium klient a Appium server

Serenity, jBehave a metody Java objektů dohromady tvoří Appium klienta. Ten pomocí knihoven vytvoří z příkazů v metodách navázaných na kroky scénářů jBehave požadavky v REST formě, které posílá Appium serveru. Server provede příkazy na testovacích zařízeních a vrátí zpět klientovi informace o výsledku provedené operace. Pokud je operace úspěšná, pokračuje Appium klient dalšími kroky testů. V případě neúspěšného provedení Appium klient ukončí komunikaci s Appium serverem, odešle informaci o selhání operace zpět nástroji jBehave, který ji zpracuje a zaznamená do svého reportu.



Obrázek 4.2: Návrh řešení – průběh automatizace

4.1.8 Appium server a testovací zařízení

Appium server přeloží požadavky od klienta v REST formátu do příkazů frameworků specifických pro jednotlivé podporované platformy. Například při testování aplikace pro Android použije příkazy frameworku UIAutomator, díky kterým může ovládat zařízení Android.

4.1.9 Jenkins a vývojář

Po skončení testů je notifikován vývojář o výsledku a je mu zprostředkován report vytvořený nástrojem Serenity (viz kapitola 4.1.4). Tato notifikace může být doručena například na email vývojáře, případně jako zpráva v interním komunikačním nástroji (například Slack) a podobně.

Kapitola 5

Implementace integrovaného systému

Na základě předchozích kapitol o výběru nástrojů a návrhu řešení implementujeme v této kapitole integrovaný systém zvolených nástrojů. Při implementaci budeme předpokládat testování platformy Android a využijeme virtualizačního nástroje *Oracle VM VirtualBox* pro virtualizaci prostředí integračního serveru. Hotové řešení je k dispozici ve formě virtuálního disku *.vhd* přiloženém na přenosném médiu k této práci.

5.1 Implementace klienta pro Appium server

Díky architektuře Appium serveru a komunikaci s klienty lze implementovat klienta pro Appium server v téměř jakémkoliv jazyce, který umožňuje ovládat síťovou komunikaci. Vzhledem k požadavku 2 a naší konfiguraci prostředí budeme náš projekt implementovat v jazyce Java.

5.1.1 Vytvoření Maven projektu

Pro správu našeho projektu je vhodné použít některý z automatizovaných nástrojů. My tedy na základě výběru v kapitole 3.6 využijeme nástroj Apache Maven (dále budu používat zkrácené označení Maven). Maven je založen na modelu *Project Object Model*, který rozšiřuje základní popis projektu zdrojovými kódy o definice závislostí na externích knihovnách a jednotlivé fáze buildů jako například spouštění testů, validace kódu a podobně. Pro popis projektu je použit zápis v jazyce XML a základní soubor obsahující popis projektu je vždy uložen v kořenovém adresáři projektu a názvem `pom.xml`. Ukázku obsahu `pom.xml` souboru nabízí příklad 5.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
            ↪ apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.xstehl16.ibp.test</groupId>
    <artifactId>testing-maven-project</artifactId>
```

```
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>

        <serenity.maven.version>1.0.47</serenity.maven.version>

    </properties>

    <dependencies>
        <dependency>
            <groupId>net.serenity-bdd</groupId>
            <artifactId>serenity-core</artifactId>
            <version>1.9.8</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>net.serenity-bdd.maven.plugins</groupId>
                <artifactId>serenity-maven-plugin</artifactId>
                <version>${serenity.maven.version}</version>
                <executions>
                    <execution>
                        <id>serenity-reports</id>
                        <phase>post-integration-test</phase>
                        <goals>
                            <goal>aggregate</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

Příklad 5.1: Obsah Project Object Model souboru pom.xml

Definice externích závislostí

Pro správnou funkčnost všech externích závislostí musíme sladit jejich verze, aby navzájem splňovaly své požadavky. Jednotlivé závislosti nástrojů získáme z jejich oficiálních dokumentací.

Knihovna Serenity-BDD a jBehave Pro náš projekt potřebujeme následující tři knihovny - *serenity-core*, která je jádrem celého Serenity frameworku, dále *serenity-junit* pro pod-

poru spouštění jednotlivých testů a knihovnu *serenity-jbehave*, která poskytuje metody pro integraci nástrojů jBehave a Serenity.

```
<dependency>
  <groupId>net.serenity-bdd</groupId>
  <artifactId>serenity-core</artifactId>
  <version>1.9.8</version>
</dependency>
<dependency>
  <groupId>net.serenity-bdd</groupId>
  <artifactId>serenity-junit</artifactId>
  <version>1.9.8</version>
</dependency>
<dependency>
  <groupId>net.serenity-bdd</groupId>
  <artifactId>serenity-jbehave</artifactId>
  <version>1.38.0</version>
</dependency>
```

Knihovna Appium java-client Oproti knihovnám pro Serenity nám pro implementaci Appium klienta stačí pouze jediná, a to *java-client*. Tato knihovna nám poskytuje veškeré potřebné metody pro spuštění a implementaci Appium testů.

```
<dependency>
  <groupId>io.appium</groupId>
  <artifactId>java-client</artifactId>
  <version>${java-client.version}</version>
</dependency>
```

Ostatní knihovny Pro správný chod testů je potřeba specifikovat i verze dalších knihoven, kupříkladu knihovnu *junit*, která musí být dle požadavků Serenity nižší než verze 5.

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.6.1</version>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.assertj</groupId>
  <artifactId>assertj-core</artifactId>
  <version>1.7.0</version>
  <scope>test</scope>
```

```
</dependency>
```

5.1.2 Struktura projektu

Implicitní struktura projektu definovaná pro Maven je následující:

```
projekt/
projekt/src/main/java
projekt/src/main/resources
projekt/src/test/java
projekt/src/test/resources
```

Předpokládejme adresář `projekt` kořenovým adresářem celého projektu. Potom adresář `projekt/src/main/java` obsahuje zpravidla definice Java tříd, které je možné kompilovat. V adresáři `projekt/src/main/resources` mohou být konfigurační a další soubory, které nejsou třídami jazyka Java. Nás ovšem převážně zajímá adresář `projekt/src/test/java` a `projekt/src/test/resources`. V těchto adresářích jsou soubory pro testování projektu. V prvním zmíněném adresáři se očekávají třídy testů a v druhém adresáři soubory s konfigurací testů a další jako například v našem případě soubory pro definici jBehave stories.

5.1.3 Testovací třídy

Adresář `projekt/src/test/java` je určen pro samotné testovací třídy. První základní třídu, ve které jsou udržovány jednotlivé kroky (*steps*) testů, nazveme `AndroidStepDefinitions`. Před samotným spuštěním testů na mobilním zařízení je nutné inicializovat WebDriver.

WebDriver

Díky integraci se Serenity-BDD můžeme inicializaci WebDriveru přenechat právě jemu. Toho dosáhneme anotováním definice WebDriveru `@Managed(driver=appium)`. Určením driveru jako Appium dáváme informaci Serenity-BDD, že chceme inicializovat AppiumDriver.

5.1.4 Kroky jBehave

Nástroj jBehave mapuje kroky ze souborů `.story` na metody testovacích tříd. Každá metoda, která má být krokem použitelných v souborech s definicí testů pomocí syntaxe jBehave (případně Gherkin), musí být anotována podle pravidel jBehave. Například chceme-li metodu `givenWidgetsComponentIsExpanded()` použít jako krok zapsaný v jBehave (Gherkin) syntaxi jako `Given widget component is expanded`, musíme tuto metodu anotovat následovně:

```
@Given("widgets component is expanded")
public void givenWidgetsComponentIsExpanded() {
        mainPage.expandWidgetComponentWithArrow();
}
```

Aby bylo možné zpřístupnit toto mapování i nástroji Maven (a dalším jako například nástroje IDE), je nutné definovat Java Embeddable třídu. Díky integraci se Serenity můžeme použít definovanou třídu `SerenityStories` a rozšířit ji třídou `AndroidStepDefinitions`.

```
public class AndroidStepDefinitions extends SerenityStories
```

Tato třída rozšiřuje třídu `JUnitStories`, která vytváří propojení mezi spouštěcím frameworkem nástroje jBehave a textovým zápisem příběhů (stories). Po spuštění projektu tedy díky třídě `SerenityStories` budou spuštěny všechny `.story` soubory nalezené v jejich defaultní nakonfigurované lokaci.

5.1.5 PageObject

Obecně při psaní kódu je vhodné vyhýbat se duplikátním blokům kódu, a předejít tak složitým opravám při sebemenší změně ve funkčnosti[29]. V případě psaní UI testů je tomu podobně. Budeme se snažit předejít definicím objektů na více místech, poněvadž při změně rozvržení UI dojde k selhání testů a nutnosti opravit všechny definice změněných UI elementů.

K tomu nám pomůžou takzvané *PageObjects* představené ve frameworku Selenium. PageObjects modelují jednotlivé testované, původně webové, stránky jako objekty. Třída rozšiřující třídu PageObject tedy definuje elementy nacházející se na stránce a akce, které s nimi lze provést. To nám přidává jistou úroveň abstrakce, kterou lze využít při definování kroků testu. Princip PageObject si můžeme uvést na příkladu.

Uvažujme stránku naší mobilní aplikace, na které chceme vykonat akci kliknutí na tlačítko označené například `button` s textem *Click me*, které otevře další stránku. Nejprve původní obrazovku definujeme třídou rozšiřující PageObject.

```
public class MainPage extends PageObject {

        @FindBy(xpath = "//android.widget.Button[@text='Click me']")
        private WebElementFacade button;

        public LoginPage(WebDriver driver) {
                super(driver);
        }

        public void accessNextPage() {
                button.click();
        }
}
```

Příklad 5.2: Třída rozšiřující PageObject

Pokud máme takto definovanou stránku jako PageObject, potom můžeme volat jednotlivé akce z třídy definující kroky jednoduše pomocí `mainpage.accessNextPage()`. Inicializace jednotlivých PageObjects opět zajistí nástroj Serenity. Serenity inicializuje třídu `AndroidStepDefinitions` s parametrem datového typu `Pages`, který je v konstruktoru přiřazen lokální proměnné stejného typu.

```
public AndroidStepDefinitions(Pages pages) {
        this.pages = pages;
}
```

Anotace `@FindBy(xpath = "//android.widget.Button[@text='Click me']")` z příkladu 5.2 specifikuje cestu v pseudo-DOM struktuře, která jednoznačně určuje element, který má být přítomen na stránce. Při multiplatformním testování lze blíže specifikovat, pro kterou platformu daná cesta platí, pro Android `@AndroidFindBy`, či pro iOS `@IOSFindBy`.

Jednotlivé třídy PageObject stránek jsou poté inicializovány za běhu testů do proměnných definovaných obdobně jako pro náš testovací příklad `private MainPage mainPage;`.

5.1.6 Nalezení identifikátorů elementů

V předchozím odstavci jsme specifikovali elementy pomoci cesty *XPath*, která se používá pro vyhledávání prvků v XML souborech[7]. K sestavení XPath můžeme použít nástroj, který nám zpřístupní elementy na obrazovce mobilního zařízení (případně emulátoru) ve struktuře XML. Takový nástroj je poskytován v oficiální distribuci Android SDK tools pod názvem *uiautomatorviewer*. Ten získá údaje o elementech zobrazených na obrazovce připojených zařízení přes ADB *(Android Debug Bridge)*[19].

5.1.7 Spuštění projektu

Maven projekt lze spustit z příkazové řádky pomocí příkazu `mvn` a následujícího výčtu cílů, které se mají spustit, a případně parametrů. Příklad spuštění projektu může vypadat následovně:

```
$ mvn clean verify -Dmetafilter="+demoapp1"
```

5.2 Konfigurace prostředí

Pro naši práci jsme zvolili operační systém Debian na základě výběru z kapitoly 3. Jak bylo zmíněno na úvod této kapitoly, prostředí pro tuto práci je virtualizované pomocí nástroje *Oracle VM VirtualBox*. Dále v této kapitole jsou popsány konfigurace jednotlivých součástí prostředí.

Java JDK

Pro správný chod nástrojů v našem prostředí budeme potřebovat v první řadě nejaktuálnější verzi Java SE Development Kit (zkráceně JDK).

```
$ sudo apt-get install default-jdk
```

Node.js

Dle požadavků dříve vybraných nástrojů (Appium 3.1) je dále nutné mít k dispozici JavaScriptové běhové prostředí NodeJS umožnující spouštění JavaScriptových programů na straně serveru. Pro instalaci NodeJS do systému Debian využijeme nástroje NVM, Node Version Manager, díky kterému můžeme jednoduše spravovat verze NodeJS dostupných v našem prostředí a předejít některým problémům s přístupovými právy po instalaci přímo z APT repozitáře (APT – Advanced Packaging Tool). Pro instalaci je v jeho repozitáři[1] nachystán instalační skript, který lze stáhnout a spustit následujícím příkazem:

```
$ curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.9/install
    ↪ .sh | bash
```

Po úspěšné instalaci můžeme přejít k samotné instalaci NodeJS. To lze učinit jednoduše příkazem:

[1] https://github.com/creationix/nvm

```
$ nvm install node
```

Spolu s Node Version Manager se nám nainstaloval i Node Package Manager, který použijeme pro následovnou instalaci Appium serveru. Můžeme ho aktualizovat příkazem (kde parametr g značí globální instalaci):

```
$ npm install -g npm
```

Android SDK

Vzhledem ke specifikaci požadavků 1, 7 a 8 musí být naše prostředí schopné zkompilovat zdrojové kódy testované aplikace do spustitelné podoby. Pro naši práci budeme počítat s kompilací zdrojových kódů aplikace pro platformu Android. K tomu je nutné mít k dispozici aktuální verzi Android SDK (z anglického *Software Development Kit*). To je standardně dodáváno spolu s produktem Android Studio, což je oficiální IDE pro Android. Nám ovšem na straně CI serveru, na kterém budeme kompilovat aplikace, stačí pouze CLI[2] verze, která obsahuje potřebné nástroje a verze SDK. Z oficiální webové stránky Android SDK[3] můžeme stáhnout tuto CLI verzi, v našem případě pro platformu Linux. Existuje i balíček z oficiálního repozitáže APT, ovšem ten neobsahuje `sdkmanager`, který se nám bude později hodit pro instalaci potřebných verzí Android SDK a akceptování jejich licenčních podmínek.

Po stažení Android SDK Tools tedy můžeme nainstalovat verzi Android SDK, kterou potřebujeme pro testování naší aplikace. Většinou nejaktuálnější verze zajistí zpětnou kompatibilitu s nižšími verzemi. V následujícím příkladu sdkmanager stáhne Android SDK 26 a aktualizuje balík platform-tools, který obsahuje nástroje pro kompilaci zdrojových kódů Android aplikace:

```
$ ./tools/bin/sdkmanager "platform-tools" "platforms;android-26"
```

Před kompilováním je nutné akceptovat licenční smlouvy Android SDK. To lze provést následujícím příkazem a dále postupovat podle pokynů vypsaných programem:

```
$ ./tools/bin/sdkmanager --licenses
```

Nastavení systémových proměnných

Pro správný chod nástrojů musíme definovat systémové proměnné pro JDK a Android SDK. Systémové proměnné je vhodné definovat vždy při přihlášení do systému, aby byly k dispozici po restartu systému. Je tedy ideální vložit jejich definice do souboru ∼`/.bash_profile`, případně jiného souboru, který je spouštěn vždy po přihlášení do systému.

```
export ANDROID_HOME=/home/xstehl16/AndroidSDK/
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

Příklad 5.3: Definice systémových proměnných

Zároveň musíme aktualizovat systémovou proměnnou `PATH` o složku JDK s binárními daty. Opět je nejvhodnější použít soubor `.bash_profile` a využít systémové proměnné definované v předchozím kroku.

```
export PATH=$PATH:$JAVA_HOME/bin
```

[2]command line interface - prostředí příkazové řádky
[3]https://developer.android.com/studio/index.html

Nástroje pro práci s Git

Aby byl náš integrační server schopný pracovat se zdrojovými kódy aplikace, které jsou dostupné z verzovacího systému, je nutné nainstalovat nástroj umožnující nám práci s verzovacím systémem Git.

```
$ sudo apt-get install git
```

Dále musíme umožnit našemu serveru bezpečně komunikovat se vzdáleným repozitářem. V této práci byla zvolena služba Bitbucket 3.5. Pro tento účel využijeme přístupu ke vzdálené službě pomocí protokolu SSH (*Secure Shell*). Na našem serveru vytvoříme privátní a veřejný klíč například pomocí programu `ssh-keygen`. Tento program obsahuje průvodce vytvořením klíče a spouští se pomocí následujícího příkazu:

```
$ ssh-keygen
```

Vygenerovaný pár veřejného a privátního klíče nám poslouží pro komunikaci s repozitářem ve službě Bitbucket.

5.3 Konfigurace vzdáleného repozitáže ve službě Bitbucket

Po vygenerování páru privátního a veřejného klíče uložíme veřejnou část na server vzdáleného repozitáře. V případě Bitbucket je tento úkon prováděný přímo v grafickém uživatelském prostředí služby. V sekci `Bitbucket Settings -> Security -> SSH keys` lze přidat veřejnou část klíče integračního serveru.

5.4 Instalace Appium serveru

Důležitou částí této práce je Appium server. Ten je implementovaný v Node.js, který byl nainstalován v předchozích krocích. Nyní tedy stačí pomocí Node Package Manager nainstalovat onen samotný Appium server.

```
$ npm install -g appium
```

Pro ověření, zda máme splněny všechny požadavky nástroje Appium, můžeme použít program `appium-doctor`. Nainstalujeme jej obdobně příkazem:

```
$ npm install -g appium-doctor
```

a spustíme:

```
$ appium-doctor
```

Pokud máme vše správně nakonfigurováno a nainstalovány požadované verze, získáme výpis podobný tomuto:

```
info AppiumDoctor Appium Doctor v.1.4.3
info AppiumDoctor ### Diagnostic starting ###
info AppiumDoctor The Node.js binary was found at: /home/xstehl16/.nvm/
    ↪ versions/node/v9.11.1/bin/node
info AppiumDoctor Node version is 9.11.1
info AppiumDoctor ANDROID_HOME is set to: /home/xstehl16/AndroidSDK/
info AppiumDoctor JAVA_HOME is set to: /usr/lib/jvm/java-8-openjdk-amd64
info AppiumDoctor adb exists at: /home/xstehl16/AndroidSDK/platform-tools/
    ↪ adb
```

```
info AppiumDoctor android exists at: /home/xstehl16/AndroidSDK/tools/
    ↪ android
info AppiumDoctor emulator exists at: /home/xstehl16/AndroidSDK/tools/
    ↪ emulator
info AppiumDoctor Bin directory of $JAVA_HOME is set
info AppiumDoctor ### Diagnostic completed, no fix needed. ###
info AppiumDoctor
info AppiumDoctor Everything looks good, bye!
```

Příklad 5.4: Výstup programu appium-doctor

Nyní máme server Appium připravený pro spuštění. Spustit ho můžeme příkazem:

```
$ appium
```

Appium server umožňuje další kofiguraci pomocí parametrů uvedených při spouštění serveru. Jako příklad lze uvést `-relaxed-security`, který od verze Appium 1.7.2 umožňuje volat vlastní příkazy na připojeném zařízení pomocí ADB (Android Debug Bridge).

5.5 Instalace a konfigurace nástroje Jenkins

Integrační nástroj Jenkins není obsažen v oficiálním repozitáři APT. Můžeme tedy buďto stáhnout připravený instalační balíček z oficiálních webových stránek nástroje Jenkins[4], nebo můžeme přidat repozitář Jenkins do konfigurace APT a poté pomocí něj Jenkins nainstalovat. V této práci zvolíme druhou možnost, a tedy přidáme klíč Jenkins repozitáře do APT konfigurace příkazy:

```
$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo
    ↪ apt-key add -
```

Dále musíme vytvořit nový záznam – soubor `.list` ve složce `/etc/apt/sources.list.d/` nazvaný například `jenkins.list` a obsahující následující řádek:

```
deb https://pkg.jenkins.io/debian-stable binary/
```

Tímto označíme cestu, ze které jsou binární soubory Jenkins přístupné programu APT. Nyní můžeme nainstalovat Jenkins klasickým způsobem, který je pro platformu Debian nejvhodnější, posloupností následujících příkazů, které nejprve aktualizují repozitáře a poté nainstalují aktuální verzi Jenkins.

```
$ apt-get update
$ apt-get install jenkins
```

5.5.1 Konfigurace serveru Jenkins

Po instalaci je Jenkins automaticky spuštěn a je přístupný lokálně. Implicitně je navázán na port 8080. Grafické uživatelské prostředí je dostupné z prohlížečů na lokální adrese `http://127.0.0.1:8080`. Po prvním spuštění se zobrazí průvodce prvotní konfigurace serveru, při níž je možné vytvořit administrátorské či uživatelské účty, zvolit instalace přídavných modulů a podobně. V první řadě je nutné nakonfigurovat globální nástroje pro správnou funkčnost Jenkins. V uživatelském prostředí v sekci *Manage Jenkins -> Global*

[4] https://pkg.jenkins.io/debian-stable/

Tool Configuration nastavíme cestu k Java JDK, který jsme nainstalovali v při konfiguraci prostředí. Dále je vhodné zkontrolovat, zda nástroj Git je správně nastaven a neobjevuje se u něj hláška o špatné konfiguraci. Další důležitou věcí je definování systémových proměnných pro Jenkins. Je třeba definovat systémovou proměnnou `ANDROID_HOME` v nastavení Jenkins - *Jenkins -> Manage Jenkins -> Configure System -> Global Properties* zaškrtnutím možnosti *Environment Variables* a jejím samotným přidáním tlačítkem *Add*.

Instalace zásuvných modulů

Pro tento projekt využijeme vybrané pluginy (zásuvné moduly) z oficiálního repozitáře Jenkins pluginů.

Android Emulator Plugin usnadňuje práci s Android emulátory. Díky tomuto pluginu je možné zvolit jako krok buildu vytvoření nového emulátoru dle zadané konfigurace nebo spuštění připraveného, již existujícího emulátoru uloženého ve formátu `.avd`.

Bitbucket Plugin umožňuje použití Bitbucket Webhook, který použijeme pro automatické spouštění Jenkins projektů při aktualizaci repozitáře.

Gradle Plugin nám umožní build projektů pro Android. Gradle a Gradle Wrapper jsou nejpoužívanější způsoby pro build projektů Android.

HTML Publisher Plugin zjednodušuje přístup k HTML souborům vygenerovaných během buildu. Pro nás bude tento plugin užitečný pro zpřístupnění Serenity reportů přímo z uživatelského prostředí Jenkins.

Parameterized Trigger Plugin nám pomůže se spouštěním dalších Jenkins projektů po skončení předchozího. Tím můžeme vytvořit řetězec Jenkins projektů, které na sebe navazují, a vytváří tak část procesu kontinuálního dodávání.

Samozřejmě dle potřeb můžeme použít i další pluginy, které nám mohou například ušetřit místo na disku *(Workspace Cleanup Plugin)* a podobně. Ty se avšak již netýkají přímo našeho projektu integrace nástrojů.

Konfigurace RSA klíče pro komunikaci se vzdáleným repozitářem

Pomocí dříve vygenerovaného RSA klíče nyní umožníme serveru Jenkins komunikovat s naším vzdáleným repozitářem ve službě Bitbucket. Nastavení přístupových údajů pro Jenkins najdeme v konfiguraci *Jenkins -> Credentials -> Add Credentials*, kde přidáme nové globální přístupové údaje pro SSH přístup s privátním klíčem. Zde zadáme cestu k našemu privátnímu klíči, uživatelské jméno, pro které je klíč validní, a případně přístupovou frázi. Takto vytvořený záznam s přístupovými údaji pomocí privátního klíče použijeme při vytváření projektů pro kompilaci mobilní aplikace a spouštění testů.

5.5.2 Vytvoření Jenkins projektu pro build mobilní aplikace

Pro vytvoření projektu pro kompilaci aplikace ze zdrojových kódů ve vzdáleném repozitáři musíme správně nakonfigurovat komunikaci Jenkins s repozitářem umístěným ve službě

Bitbucket. V předchozím kroku jsme konfigurovali přístupové údaje s privátním klíčem, které nyní aplikujeme. Vytvoříme tedy nový projekt *Jenkins -> New item -> Freestyle project* a vyplníme základní položky (název, popis a podobně) podle potřeb.

Dále specifikujeme jako *Source Code Management* nástroj *Git*. V něm vyplníme údaje pro vzdálený přístup k repozitáři protokolem SSH. Zvolíme název větve repozitáře, ze které chceme získávat zdrojové kódy aplikace.

V sekci *Build triggers* zaškrtneme položku *Build when change is pushed to Bitbucket*. Tímto zajistíme, že projekt bude očekávat zprávu od Bitbucket Webhook, kdykoliv v repozitáři vznikne nová změna. URL, na které Jenkins očekává požadavek typu POST, je následovná:

```
http://<jenkins-url>/bitbucket-hook/
```

Vzhledem k tomu, že v našem případě se zabýváme testováním aplikace pro platformu Android, přidáme krok projektu *Build -> Add build step -> Invoke gradle step*, vybereme *Gradle Wrapper* a zadáme požadované názvy úkolů, které vedou ke kompilaci aplikace a vygenerování *APK* souborů. Gradle Wrapper nás oprošťuje od nutnosti manuálně instalovat nástroj Gradle do našeho systému a zároveň zaručuje spuštění té verze Gradle, pro kterou je build navržen.

5.5.3 Vytvoření Jenkins projektu pro spuštění testů

Pro vytvoření projektu[5] pro spuštění samotných testů nad aktuální verzí mobilní aplikace zkompilované v projektu pro build mobilní aplikace můžeme postupovat obdobně jako v odstavci 5.5.2. Nastavíme přístup k repozitáři obsahujícím zdrojové kódy našeho projektu s UI testy. Pro splnění požadavku 8 nastavíme spouštěč testů, aby reagoval na úspěšné dokončení projektu pro build mobilní aplikace. Jako krok buildu zvolíme volání Maven cílů – *Invoke top-level Maven targets* a přidáme cíle, kterými spustíme Maven projekt – *clean* a *verify*. Parametry můžeme předat konfigurační data pro Appium server. Například můžeme zadat vzdálenou cestu pro přístup k vygenerovaným souborům APK. Ten získáme buďto přímo v projektu pro build aplikace, nebo ho můžeme kopírovat do jiné lokace v projektu pro spouštění testů. Příklad lokální cesty k vygenerovanému APK souboru může být:

```
http://localhost:8080/job/build-android-app/ws/app/build/outputs/apk/debug/
    ↪ app-debug.apk
```

Díky možnosti předávat konfiguraci serveru Appium pomocí parametrů Maven projektu můžeme docílit multiplatformního testování tak, že pro spouštění testů na jiných platformách zadáme potřebné údaje v každém projektu zvlášť. Není tedy nutné řešit vícero konfiguračních souborů pro nástroj Serenity a server Appium.

Po dokončení buildu můžeme ještě specifikovat další úkony, které se provedou nezávisle na výsledku buildu. Toho využijeme pro archivaci a zpřístupnění reportů vygenerovaných nástrojem Serenity za pomoci Jenkins pluginu *HTML Publisher*. Ten vytvoří odkaz v uživatelském prostředí Jenkins na projektové i buildové úrovni a archivuje soubory reportu.

Dalším užitečným úkonem může být notifikace členů týmu o dokončení běhu testů. Lze přidat například emailovou notifikaci.

[5]Jenkins projekt definuje kroky buildu a konfiguraci prostředí, ve kterém je tento build spouštěn

Kapitola 6

Ověření funkčnosti systému

Pro ověření, zda námi implementovaný systém funguje, je vytvořena příkladová aplikace pro platformu Android. Na ní jsou ověřeny možnosti systému, jeho nedostatky a možnosti zlepšení.

Ve druhé fázi je ověřena funkčnost systému s použitím reálné aplikace dostupné z distribučních služeb.

Nakonec je systém otestován při nasazení do reálného prostředí vývojářské firmy.

6.1 Příkladové aplikace

Na příkladové aplikaci jsou ověřeny tři části systému – *automatické spuštění buildů a testování po aktualizaci repozitáře*, *samotné testování UI aplikace* a nakonec *generování reportů.*

6.1.1 Průběh ověření

Byla vytvořena jednoduchá aplikace nazvaná `DemoApp1` vycházející z oficiálních příkladů dodávaných s nástrojem Android Studio. Tato aplikace se skládá ze dvou aktivit[1] – `LoginActivity` a `MainActivity`. V aktivitě `LoginActivity` je jednoduchý formulář, který slouží k přihlášení. V aktivitě `MainActivity` se zobrazí text s informací o úspěšném přihlášení. Pokud se přihlášení nezdaří, uživatel zůstává v aktivitě `LoginActivity` a má možnost opakovat pokus. Validní přihlašovací údaje jsou `user/user`, kdy první je přihlašovací jméno a druhé heslo. Zdrojové kódy aplikace jsou uchovávány v repozitáři ve službě Bitbucket. Při každé změně ve vzdáleném repozitáři je spuštěn Bitbucket Webhook, který posílá požadavek na adresu Jenkins serveru, který ho zpracuje a spouští *build.* Tento build stáhne aktuální repozitář, zkompiluje aplikaci a spustí UI testy. Ty použijí vygenerovaný APK soubor k instalaci aplikace na reálné zařízení, na kterém proběhnou samotné testy.

6.1.2 Výsledek ověření

Vzhledem k tomu, že je práce implementována ve virtuálním systému spuštěném virtualizačním nástrojem *Oracle VM VirtualBox*, není vzhledem k hardwaru, na kterém tento nástroj spouštím, možné vytvořit emulátor Android zařízení. Lze tedy ověřit pouze spouštění testů na reálném zařízení.

[1] Aktivita představuje jednotlivou komponentu Android apliakce, která definuje grafické uživatelské prostředí a funkcionalitu aktivity[18].

Dále vzhledem k vývoji v lokálním prostředí, kdy server Jenkins není dostupný z vnější sítě, není možné otestovat funkčnost Bitbucket Webhook. Při ověřování byl tedy spuštěn build manuálně po aktualizaci repozitáře. Přesto je Jenkins projekt pro build aplikace nakonfigurovaný ke spouštění pomocí Bitbucket Webhook a v případě nasazení do reálného prostředí stačí v repozitáři Bitbucket nakonfigurovat správnou adresu Jenkins, na které očekává zprávu o aktualizaci repozitáře.

Samotný průběh integrace proběhl bezproblémově, Jenkins projekt spouštějící build aplikace po jeho úspěšném dokončení vyvolal spuštění projektu s UI testy, které úspěšně proběhly na reálném zařízení a následně byl vygenerován očekávaný report dostupný z webového prostředí Jenkins.

6.2 Reálná aplikace

Pro celkové ověření funkčnosti systému byla zvolena aplikace ČSFD[2], která je dostupná z oficiální distribuční služby Google Play. Aplikace ČSFD poskytuje katalog filmů, vyhledávání v nich, vytváření vlastních seznamů, možnost přidat komentář k filmu a podobně.

6.2.1 Průběh ověření

Vzhledem k tomu, že aplikace je získávána z oficiální distribuční služby Google Play Store, nemáme tedy přístup ke zdrojovým kódům ani instalačním balíčkům.

Cílem je ověřit spouštění automatického testování i s aplikacemi, které jsou přítomny na testovacím zařízení před spuštěním testů. Test pro ověření provede vyhledání filmu *Blues Brothers* a zkontroluje, zda byl film vyhledán a zařazen mezi výsledky vyhledávání. Dalším testem bude vyhledání neexistujícího názvu filmu a kontrola, zda aplikace notifikovala uživatele o neexistenci takového filmu v databázi.

6.2.2 Výsledek ověření

Výsledek byl opět pozitivní, podobně jako u příkladové aplikace nebylo možné spustit testy na emulátoru, a bylo tedy využito reálné zařízení. Pro ilustraci bylo do testu zařazeno i selhání kroku, které je následně správně vygenerováno ve výsledném reportu.

6.3 Reálné nasazení

Vzhledem k požadavku 11 na aplikaci systému do reálného prostředí byl tento systém otestován nasazením ve firmě, která se zčásti zabývá tvorbou mobilních aplikací. Při nasazení byla ověřena kompletní funkcionalita automatizace od nahrání změn do vzdáleného repozitáře vývojářem až po vygenerování reportu testu.

[2]https://play.google.com/store/apps/details?id=cz.csfd.csfdroid&hl=cs

Kapitola 7

Závěr

V práci byly analyzovány dostupné nástroje pro podporu automatizace testování mobilních aplikací a ověřeny možnosti jejich integrace pro vytvoření uceleného automatizovaného testovacího cyklu, který nevyžaduje manuální činnost testera. Díky použitým nástrojům je možné dále rozšířit implementovaný systém o podporu jiných mobilních platforem, jakými jsou iOS nebo UWP.

Díky využití nástroje pro podporu kontinuální integrace je možné rozšířit automatizaci o další kroky jako například statickou či dynamickou analýzu kódu použitím automatizovaného nástroje MobSF[1]. Tím lze automatizovaně poskytnout vývojáři informace o stavu a kvalitě zdrojových kódu. Je také možné provést propojení s dalšími službami pro Quality Assurance (například TestRail), integraci do procesu kontinuálního dodávání a podobně. Možností je opravdu mnoho a díky funkčním komunitám těchto open-source nástrojů se možnosti neustále rozšiřují.

Architektura nástroje Appium zase umožňuje vzdálené testování, a je tedy možné samotné spouštění testů přenechat specializovaným službám jako *SauceLab* s tím, že ostatní kroky integrace zůstávají na naší straně. Můžeme tak zefektivnit výdaje na vlastní testovací zařízení a hardware potřebný pro testovací server a sestavování (například testy iOS aplikace musí být spouštěny na zařízení s operačním systémem OS X schopné zkompilovat IPA soubor).

[1] Mobile Security Framework - https://github.com/MobSF/Mobile-Security-Framework-MobSF

Literatura

[1] Android: *Espresso*. [Online; navštíveno 06.05.2018].
URL https://developer.android.com/training/testing/espresso/

[2] Apple Inc: *Why Objective-C?* [Online; navštíveno 06.05.2018].
URL https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/OOP_ObjC/Articles/ooWhy.html

[3] Apple Inc.: *XCTest*. [Online; navštíveno 06.05.2018].
URL https://developer.apple.com/documentation/xctest

[4] Atlassian Corporation Plc: *Get started with Bitbucket Pipelines*. [Online; navštíveno 06.05.2018].
URL https://confluence.atlassian.com/bitbucket/get-started-with-bitbucket-pipelines-792298921.html

[5] Atlassian Corporation Plc: *Understanding the Bamboo CI Server*. [Online; navštíveno 06.05.2018].
URL https://confluence.atlassian.com/bamboo/understanding-the-bamboo-ci-server-289277285.html

[6] Balog, M.; Gaunt, A. L.; Brockschmidt, M.; aj.: *DeepCoder – Learning to Write Programs*. 2016, [Online; navštíveno 03.05.2018].
URL https://arxiv.org/abs/1611.01989

[7] Bradley, N.: *XML – kompletní průvodce*. Grada, 2000, ISBN 80-7169-949-7.

[8] Braun, S.; Elberzhager, F.; Holl, K.: *Automation Support for Mobile App Quality Assurance — A Tool Landscape*. [Online; navštíveno 03.05.2018].
URL https://www-sciencedirect-com.ezproxy.lib.vutbr.cz/science/article/pii/S187705091731308X

[9] Burd, B. A.: *Java Programming for Android Developers for Dummies*. For Dummies, 2016, ISBN 1119301084.

[10] Colantonio, J.: *BDD — Serenity now! Thucydides is now Serenity*. [Online; navštíveno 06.05.2018].
URL https://www.joecolantonio.com/2014/12/02/bdd-serenity-now-thucydides-is-now-serenity/

[11] Cucumber.io: *Executable Specifications*. [Online; navštíveno 06.05.2018].
URL https://docs.cucumber.io/guides/overview/

[12] Davies, A.: *Continuous Integration: Hudson vs. Jenkins*. [Online; navštíveno 06.05.2018].
URL https://www.devteam.space/blog/continuous-integration-hudson-vs-jenkins/

[13] Fowler, M.: *BusinessReadableDSL*. [Online; navštíveno 06.05.2018].
URL https://martinfowler.com/bliki/BusinessReadableDSL.html

[14] Frank: *Testing With Frank*. [Online; navštíveno 06.05.2018].
URL http://testingwithfrank.github.io/

[15] García, C. G.; Espada, J. P.; Bustelo, B. C. P. G.; aj.: *Swift vs. Objective-C – A New Programming Language*. 2015, [Online; navštíveno 03.05.2018].
URL https://dialnet.unirioja.es/servlet/articulo?codigo=5574309

[16] GitLab: *GitLab Features*. [Online; navštíveno 06.05.2018].
URL https://about.gitlab.com/features/

[17] Google Inc.: *EarlGrey Reference*. [Online; navštíveno 06.05.2018].
URL http://google.github.io/EarlGrey/

[18] Google Inc.: *Introduction to Activities*. [Online; navštíveno 06.05.2018].
URL https://developer.android.com/guide/components/activities/intro-activities

[19] Google Inc.: *UI Automator*. [Online; navštíveno 06.05.2018].
URL https://developer.android.com/training/testing/ui-automator

[20] Hans, M.: *Appium Essentials*. Packt Publishing, 2015, ISBN 1784392480.

[21] Hao, S.; Liu, B.; Nath, S.; aj.: *PUMA: Programmable UI-Automation for Large-Scale Dynamic Analysis of Mobile Apps*. [Online; navštíveno 03.05.2018].
URL https://www.sigmobile.org/mobisys/2014/pdfMainConference/sys269-hao.pdf

[22] Humble, J.: *Continuous delivery*. Addison -Wesley, 2011, ISBN 9780321601919.

[23] ios-driver: *ios-driver*. [Online; navštíveno 06.05.2018].
URL https://ios-driver.github.io/ios-driver/

[24] jBehave: *What is JBehave?* [Online; navštíveno 06.05.2018].
URL http://jbehave.org/index.html

[25] jBehave: *What is jBehave? - Story Syntax*. [Online; navštíveno 06.05.2018].
URL http://jbehave.org/reference/latest/story-syntax.html

[26] Jenkins.io: *Jenkins User Documentation*. [Online; navštíveno 06.05.2018].
URL https://jenkins.io/doc/

[27] JetBrains s.r.o: *TeamCity Features*. [Online; navštíveno 06.05.2018].
URL https://www.jetbrains.com/teamcity/features/

[28] Kudryashov, K.: *Behat documentation*. [Online; navštíveno 06.05.2018].
URL http://behat.org/en/v2.5/

[29] Martin, R. C.: *Clean code – a handbook of agile software craftmanship*. Prentice Hall, 2009, ISBN 978-0-13-235088-4.

[30] Meyer, B.: *Agile! – the good, the hype and the ugly*. Springer, 2014, ISBN 978-3-319-05154-3.

[31] North, D.: *Introducing BDD*. [Online; navštíveno 06.05.2018].
URL https://dannorth.net/introducing-bdd/

[32] Pittet, S.: *Continuous integration vs. continuous delivery vs. continuous deployment*. [Online; navštíveno 03.05.2018].
URL https://www.atlassian.com/continuous-delivery/ci-vs-ci-vs-cd

[33] Polo, M.; Reales, P.; Piattini, M.; aj.: *Test Automation*. [Online; navštíveno 03.05.2018].
URL https://ieeexplore-ieee-org.ezproxy.lib.vutbr.cz/document/6401116/

[34] Ramos, M.: *Continuous Integration, Delivery, and Deployment with GitLab*. [Online; navštíveno 06.05.2018].
URL https://about.gitlab.com/2016/08/05/continuous-integration-delivery-and-deployment-with-gitlab/

[35] Selendroid.io: *Selendroid*. [Online; navštíveno 06.05.2018].
URL http://selendroid.io/

[36] Singh, S.; Gadgil, R.; Chudgor, A.: *Automated Testing of Mobile Applications using Scripting Technique: A Study on Appium*. [Online; navštíveno 03.05.2018].
URL
http://inpressco.com/wp-content/uploads/2014/10/Paper973627-3630.pdf

[37] Singh, V.: *Gherkin*. [Online; navštíveno 06.05.2018].
URL http://toolsqa.com/cucumber/gherkin/

[38] Smart, J. F.: *The Serenity Reference Manual*. [Online; navštíveno 06.05.2018].
URL http://thucydides.info/docs/serenity-staging/

[39] Software in the Public Interest, Inc. and others: *About Debian*. [Online; navštíveno 06.05.2018].
URL https://www.debian.org/intro/about

[40] SpecFlow.org: *SpecFlow*. [Online; navštíveno 06.05.2018].
URL http://specflow.org/

[41] Travis CI GmbH: *Travis CI*. [Online; navštíveno 06.05.2018].
URL https://travis-ci.com/

[42] Xamarin: *Introduction to Calabash*. [Online; navštíveno 06.05.2018].
URL https://developer.xamarin.com/guides/testcloud/calabash/introduction-to-calabash/

[43] Zadgaonkar, H.: *Robotium Automated Testing for Android*. Packt Publishing, 2013, ISBN 9781782168027.

Příloha A

Spuštění integrovaného systému

Na přiloženém médiu se nachází obraz virtuálního disku ve formátu Virtual Hard Disk s příponou `.vhd`, který obsahuje obraz systému Debian, v němž je implementován integrovaný systém.

Pro spuštění obrazu je nutné použít virtualizační nástroj – například *Oracle VM VirtualBox*. Po přihlášení do systému se automaticky spustí server Jenkins. Pro přístup do jeho administrace lze použít předinstalovaný prohlížeč Firefox. Jenkins je dostupný na adrese `localhost:8080`.

Před spuštěním integrace je nutné připojit reálné Android zařízení a spustit Appium server příkazem `appium`. Spuštěním Jenkins projektu `build-demoapp1` se zahají integrace nástrojů.

Pro spuštění testů s reálnou aplikací CSFD je nutné ji nejdříve nainstalovat na testovací zařízení a poté spustit projekt mobile-ui-tests s parametry CSFD (vždy druhá možnost v nabídce parametru).

Přístupové údaje jsou přiloženy na paměťovém médiu ve složce `prihlasovaci-udaje` a pro každý přístup v odpovídajícím souboru (například údaje pro přihlášení do sytému Debian jsou v souboru `debian.txt`).