FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Automatizace procesů vývoje mobilních aplikací
Student:	Tomáš Lála
Vedoucí:	Ing. Michal Czinner
Studijní program:	Informatika
Studijní obor:	Informační systémy a management
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

1) Seznamte se s mobilním vývojem a jeho procesy.
2) Analyzujte aktuálně dostupné služby pro automatizaci procesů. Služby k analýze vyberte po dohodě s vedoucím práce.
3) Na základně analýzy navrhněte řešení, které podporuje všechny procesy mobilního vývoje. Navržené řešení poté zrealizujte pro mobilní platformu Android.
4) Popište, jaký byl pracovní postup projektového týmu před a po nasazení automatizace procesů.
5) Analyzujte a uveďte přínosy automatizace procesů pro projektový tým. Přínosy poté porovnejte s náklady na automatizaci procesů pro střední firmu (50 uživatelů, 25 aplikací).

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 21. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Automatizace procesů vývoje mobilních aplikací

Tomáš Lála

Katedra softwarového inženýrství
Vedoucí práce: Ing. Michal Czinner

13. května 2018

Poděkování

Děkuji panu Ing. Michalu Czinnerovi za vedení této práce, dále také panu Jiřímu Pechovi za velmi cenné rady a informace v průběhu práce. Taktéž děkuji své rodině za podporu během celého studia.

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 13. května 2018 .....

Odkaz na tuto práci

Lála, Tomáš. *Automatizace procesů vývoje mobilních aplikací.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce se zabývá analýzou a realizací služeb pro automatizaci procesů. Součástí bakalářské práce je identifikace jednotlivých procesů v životním cyklu vývoje mobilní aplikace. Práce představuje i pracovní postup, který standardizuje procesy projektového týmu tak, aby na ně bylo možné navázat automatizovanými procesy. Na základě stanoveného postupu a vybraných služeb je realizováno řešení pro Android aplikaci. Následně tato práce obsahuje přínosy a náklady, které vybrané řešení přináší.

Klíčová slova automatizace procesů, mobilní vývoj, pracovní postup, automatizované testy, Android, iOS

Abstract

This bachelor thesis is focused on analysis and realization of services for process automation. The bachelor thesis firstly contains identifying individual processes in the lifecycle of mobile application development. In addition, this thesis represents Workflow which standardizes the processes of the project team so that they can be connected by automated processes. Based on Workflow and selected services, an Android application is implemented. Finally, this thesis concerns the benefits and costs that the chosen solution brings.

Keywords process automation, mobile development, Workflow, automated tests, Android, iOS

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Vývoj programů se v posledních patnácti letech dost razantně změnil. Současné programy obsahují statisíce řádků kódu, implementují desítky knihoven a jsou integrovány s dalšími programy a systémy. Pro udržení kvality programů jsou klíčové nové procesy, které zajišťují stabilitu, bezpečnost, rozšiřitelnost a udržovatelnost programu. Pokud jsou tyto procesy prováděny manuálně, jsou časově a finančně náročné. Proto je snaha tyto procesy automatizovat.

V první části této práce popisuji mobilní vývoj a jeho specifika. Zároveň identifikuji základní procesy životního cyklu vývoje. V následující části jsou poté analyzovány služby a programy pro jejich automatizaci. Při nasazení automatizace procesů se zároveň mění i pracovní postup vývojového týmu. Této problematice se věnuji ve třetí kapitole své práce. Z vybraných služeb a nadefinovaných postupů je poté sestaveno řešení, které je implementováno pro mobilní aplikaci systému Android. V závěru práce vyhodnocuji přínosy a náklady na automatizaci procesů.

Téma své bakalářské práce jsem si zvolil se záměrem zmapovat postup, výběr a nasazení automatizace pro mobilní vývoj. Výběr služeb, realizace řešení a vyhodnocení nákladů a přínosů je uzpůsobeno pro firmu o padesáti zaměstnancích, která spravuje dvacet pět aplikací. Firma o padesáti zaměstnancích je právě na hraně mezi malým a středním podnikem. Při takovém počtu zaměstnanců již vzniká potřeba procesy standardizovat.

Kapitola 1

Mobilní vývoj a jeho procesy

V současnosti není mobilní vývoj pouze o vytváření aplikací pro chytré telefony. Operační systémy, které byly z počátku zamýšleny pro telefony, nyní pohání zařízení jako tablety, chytré hodinky, a dokonce i televize. Procesy ve vývoji mobilních aplikací se od procesů vývoje pro desktopová či webová zařízení příliš neliší. Přesto existují odlišnosti, které je třeba mít na zřeteli.

První z nich je volba podporovaných operačních systémů. V práci se zabývám pouze operačními systémy Android a iOS, jelikož jejich zastoupení na světovém trhu tvoří dohromady 94.93 % [1]. Donedávna, kromě těchto dvou systémů, firmy vyvíjely aplikace i pro Windows Phone. Zařízení s tímto systémem celosvětově pokrývá pouze 0.5 %. Zároveň aktivní vývoj pro tento systém byl ukončen, a proto není součástí této práce [2].

Další odlišností je to, že mobilní aplikace nejsou určeny pro složité výpočetní činnosti. Důvodem je, že součástky zařízení nesmí přesahovat velikostní rozměry a jejich chlazení je taktéž značně omezeno. Velmi k tomu přispívá i fakt, že zařízení není neustále připojeno k elektrické síti, ale musí fungovat na baterii. Právě kapacita baterie je jedním z největších problému mobilních zařízení.

Při grafickém návrhu je nutné dbát na to, že velikost zobrazovací plochy mobilního zařízení je od počítačového monitoru násobně menší. Proto firmy vyvíjející operační systémy vytváří doporučující návody pro správný návrh. Tyto návody definují, jak mají grafické prvky vypadat, kde mají být umístěny, a jak by se měly chovat. Pravidla v návodech se snaží vývojáři dodržovat a díky tomu uživatelé nemusí u každé aplikace zjišťovat, jak se v ní pohybovat. Při nedodržení daných návodů je zvýšené riziko, že uživatel bude v aplikaci tápat a nebude ji používat. To může vést ke špatnému hodnocení aplikace na distribučních serverech a následnému snížení počtu nových instalací. Problém je, že pro každý operační systém jsou vytvářeny odlišné grafické návody [3].

1.1 Android

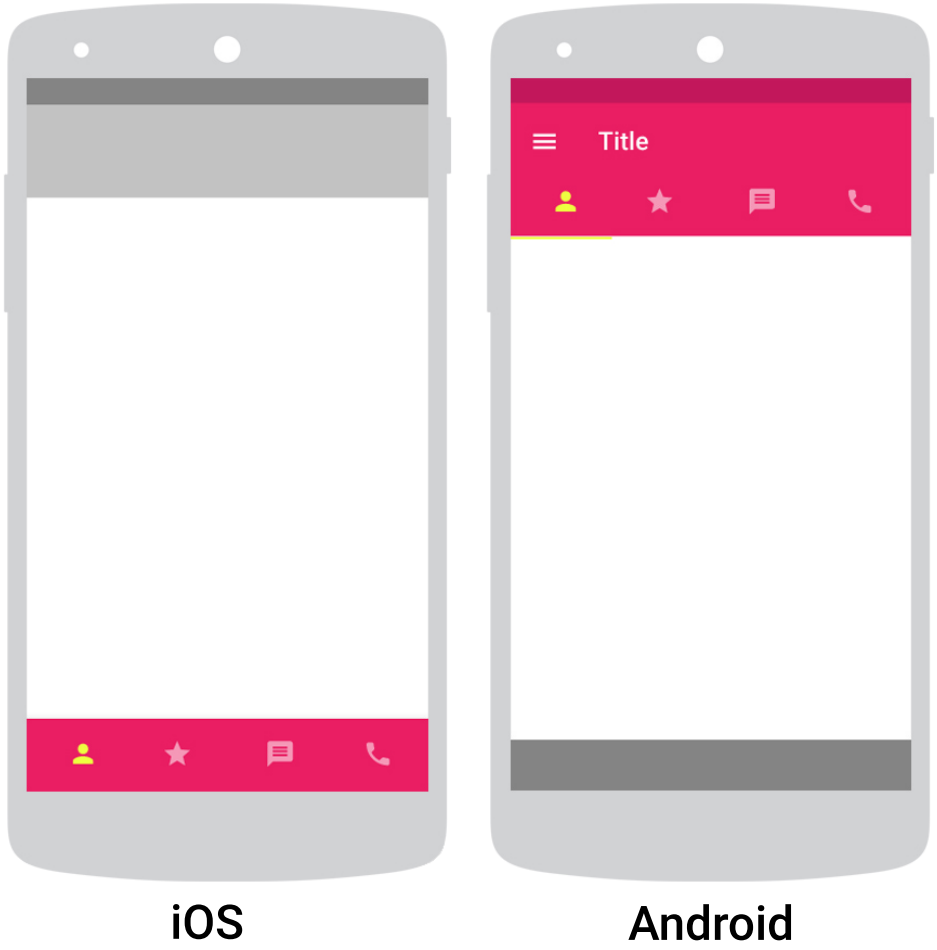
Android je operační systém založený na linuxovém jádře [4]. Historicky první verze vyšla v roce 2008 a dnes, tedy o deset let později, je již ve vývoji 9. verze tohoto systému [5]. Systém Android je nejrozšířenější operační systém na světě a jedním z důvodů může být to, že je téměř od samého začátku open-source [6]. Firmy vyvíjející mobilní zařízení po hardwarové stránce nemusí vytvářet nové operační systémy, ale mohou využít Android, který dle potřeby upraví pro svá zařízení. Primárně tento systém vyvíjí a udržuje společnost Google. Systémy jako Wear OS nebo Android Things staví právě na systému Android a jsou pouze upraveny pro lepší kompatibilitu se zařízením. Od počátku byl jediným nativním programovacím jazykem Java, ke kterému v roce 2017 přibyl jazyk Kotlin [7].

1.2 iOS

Operační systém iOS je postaven na jádře XNU. Jak tento systém, tak i jádro vyvíjí firma Apple. První verze systému byla představena v roce 2007 s prvním chytrým telefonem od společnosti Apple. Důležitým bodem pro automatizaci je, že iOS aplikace se dají sestavovat jen s iOS SDK. SDK obsahuje jednotlivé funkce systému, které lze v aplikaci implementovat. Toto SDK lze naistalovat pouze na operační systém macOS, který je taktéž vyvíjen společností Apple. Dle EULA Applu není legální virtualizovat či instalovat macOS na jiné zařízení než od Applu [8]. Proto, aby iOS aplikace mohla být legálně sestavena, je nutné využívat pouze zařízení vyrobené společností Apple. Narozdíl od Android je iOS uzavřen a není možné tento systém volně instalovat na jiná zařízení. Nativní programovací jazyk je Swift, který nahradil předtím využívaný Objective-C.

1.3 Nativní vs. multiplatformní vývoj

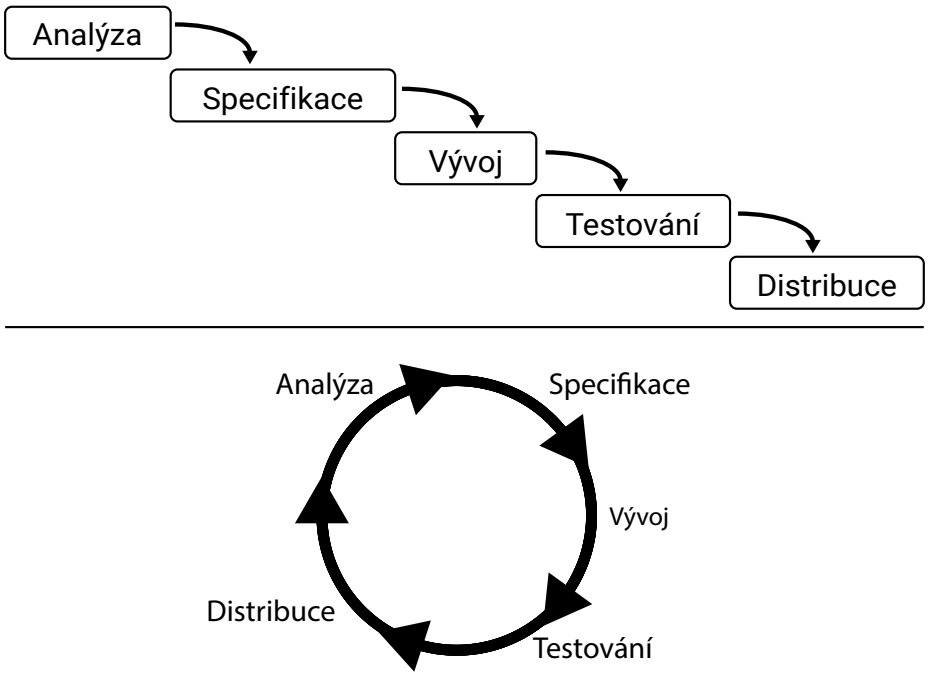
Při nativním vývoji je aplikace napsána v programovacím jazyce, který je firmou vyvíjející operační systém plně podporován. U multiplatformního vývoje je programovací jazyk zvolen jiný než nativní a pomocí různých nástrojů a modifikací při sestavování aplikace upraven tak, aby byl spustitelný na více operačních systémech zároveň. Multiplatformní vývoj tedy s sebou nese tu výhodu, že není nutné psát celou aplikaci vícekrát v různých jazycích, ale pouze jednou s možnými úpravami pro jednotlivé systémy. To se ale nakonec může stát nevýhodou, když systémy nepodporují stejné funkce, nebo je nutné tyto funkce implementovat odlišně pro každý systém. Na obrázku 1.1 je možné vidět stejné menu, kterým lze přecházet mezi obrazovkami aplikace, ale jejich umístění a styl jsou odlišné. Důvodem je to, že pro oba systémy jsou nastaveny jiné grafické návody.



Obrázek 1.1: Rozdíl Android a iOS grafických návodů

1.4 Procesy

V životním cyklu vývoje softwaru existuje pět základním procesů. Pro dodání kvalitního softwaru není možné, aby byl jeden, nebo více procesů z životního cyklu vynechány. Různé metodiky řízení vývoje mohou tyto procesy jinak pojmenovávat, nebo je slučovat a rozdělovat. Obecně platí, že tyto procesy mají takové pořadí, které je dodrženo i v této kapitole. Metodika s názvem Waterfall toto pořadí striktně dodržuje a není možné procesy znovu opakovat. Opakem jsou iterativní metodiky, kde se procesy opakují a umožňují běh více procesů najednou. V této práci se primárně zaměřuji na rozpad procesů týkajících se vývoje, testování a distribuce. Právě tyto procesy se ve vývoji automatizují, jelikož většinu z nich provádí stroj.



Obrázek 1.2: Waterfall vs. iterativní vývoj

1.4.1 Analýza

Analýza je první proces, při kterém probíhá sběr požadavků od zainteresovaných stran na aplikaci. Požadavky se dělí do dvou typů. Prvním typem jsou funkční požadavky, které definují chování aplikace. Druhý typem jsou požadavky nefunkční, ty definují potřeby na prostředí či kontext, kde aplikace bude fungovat. Zároveň u některých požadavků probíhá prototypizace. Do tohoto procesu se mnohdy nevkládá mnoho úsilí, jelikož tento proces nevytváří finální produkt. Na druhou stranu při dobré analýze je možné najít problémy, které by se objevily až ve vývojovém procesu. V lepším případě by objevené problémy bylo možné opravit, ale samozřejmě na úkor času a navýšení nákladů. V horším případě jsou problémy tak závažné, že požadavku vůbec nelze docílit z důvodu, že neproběhla potřebná analýza či prototypizace.

1.4.2 Specifikace

V tomto procesu probíhá vytváření dokumentů, které popisují aplikaci jak z pohledu vývoje, tak i z pohledu uživatele. Dokumenty obsahují návrh řešení aplikace na základě všech zjištěných požadavků a informací z předešlého

procesu. Zároveň s dokumenty vznikají i wireframy, které popisují chování jednotlivých obrazovek aplikace a jejich přechody mezi nimi. Na základě wireframů a dokumentů vzniká grafický design, který je obvykle vytvářen pro každý systém zvlášť. Wireframy, grafický design a dokumentace dohromady tvoří potřebné informace pro vývoj aplikace a její testování.

1.4.3 Vývoj

Ve vývojovém procesu je implementován zdrojový kód dle specifikací a grafického návrhu. Primárně je tento proces v samotném životním cyklu vývoje nejdelší.

V základu je možné sestavit mobilní aplikaci ve dvou variantách. První varianta je určena pro vývoj a testování díky tomu, že umožňuje jednodušší získávání informací o chybách aplikace. Druhá varianta je určena pro koncové uživatele, která zabraňuje možnost záměny aplikace za jinou. K těmto dvěma variantám lze přidávat další a zároveň je mezi sebou kombinovat. Toho se využívá například, když aplikace pracuje s daty, která získává z webových stránek. Pro vývoj a testování se tedy využijí webové stránky, které neobsahují reálná data uživatelů, ale testovací, která lze dle potřeby měnit. Poté pro koncové uživatele samozřejmě aplikace musí komunikovat se stránkou, která je spjatá s reálnými daty.

Pro možnost testovaní aplikace na zařízeních je nutné vytvořit instalační balíčky, které jsou poté do nich instalovány. A právě tento proces, lze plně automatizovat. Tedy stroj dle pravidel, která jsou definována v kapitole 3, sestaví ze zdrojových kódů instalační balíček mobilní aplikace ve správné variantě, který potom předá dál k následnému zpracování.

1.4.4 Testování

1.4.4.1 Uživatelské testování

Tento druh testování je nejpoužívanější, jelikož odhaluje funkční i nefunkční nedostatky aplikace, jako její pády, špatné umístění grafických prvků a nesprávné chování aplikace. Toto testování využívají vývojáři pro rychlé zjištění, zda se zdrojový kód chová tak, jak očekávají. Dále ho využívají grafici, aby zjistili, zda jimi navržené uspořádání obrazovky je dostatečně intuitivní pro uživatele a testeři, kteří testují podrobně chování aplikace. Testování nelze plně nahradit jinými automatizovanými testy, protože je prováděno člověkem.

Pro kompletní pokrytí aplikace uživatelskými testy vznikají testovací scénáře, které popisují jednotlivé kroky, jak aplikaci dostat do určitého stavu. Po úspěšném splnění všech kroků ve scénáři je stav aplikace porovnán s tím, co bylo specifikováno v dokumentaci. Otestování všech scénářů u větší aplikace může při manuální práci zabrat i několik hodin. Proto vznikají další druhy testů, které právě snižují potřebu vše uživatelsky testovat. Nikdy ale nelze

uživatelské testování plně nahradit a je nutné při automatizaci procesů s ním počítat.

1.4.4.2 Jednotkové testování

Pro objektově orientované jazyky se jedná o testování tříd nebo metod. Testy jsou psány ve zdrojovém kódu většinou vývojářem, který danou třídu nebo metodu vytvořil. Jednotkový test by měl ověřit, zda se daná část kódu chová korektně pro všechny možné stavy, které mohou nastat. Tento druh testu netestuje chování mobilní aplikace jako celku, ale pouze její části, ke kterým není potřeba reálné či simulované zařízení. Abychom tedy danou funkcionalitu mohli otestovat, je nutné ji řádně oddělit tak, aby nebylo třeba využívat kód, který se stará o běh aplikace. Proto ve většině případů nelze tento druh testů nasadit v průběhu vývoje. Pokud bude při vývoji jednotkové testování využíváno, je nezbytné na tento fakt pamatovat již při návrhu struktury aplikace [9].

1.4.4.3 UI testování

UI testování na rozdíl od jednotkového probíhá na reálném zařízení, nebo simulátoru. Tento druh testů umožňuje ověřovat několik funkcí v aplikaci. Nejpoužívanějším způsobem je ověření, zda aplikace správně reaguje na vstupy uživatele, nebo stavy aplikace. UI testování se taktéž využívá pro ověření, zda průchod aplikací umožňuje uživateli dostat se na všechny obrazovky. Společná kombinace těchto způsobů tvoří možnost simulovat testovací scénáře z 1.4.4.1. Pro vytvoření těchto testů se používají frameworky, které umožňují simulovat uživatele a umí pracovat s životním cyklem mobilní aplikace. UI testy již mohou psát testeři, kteří mají základy programování a znají syntaxi jazyka, ve kterém se testy píší.

1.4.4.4 Statická analýza kódu

Statická analýza kódu nekontroluje funkčnost aplikace tak, jako předešlé testy, ale hledá chyby ve zdrojovém kódu. Pro spuštění analýzy tedy není nutný běh aplikace. Většina moderních IDE již má základní analýzu kódu v sobě zabudovanou. Analýza pracuje na principu prohledávání kódu a hledání chybových vzorů jako je duplicita kódu, nedosažitelné části kódu, nezachycení výjimek, nebo jmenné konvence. [10] Vzhledem k tomu, že programovací jazyky mají různé konvence a zápisy kódu, znamená to, že i statická analýza musí být vytvořena vždy specificky pro konkrétní jazyk a jeho verzi.

1.4.5 Distribuce

Ve chvíli, kdy je aplikace sestavena a otestována, přichází na řadu poslední proces ve vývojovém cyklu, který se nazývá distribuce. Obecně se tento pro-

ces označuje jako nasazení, ale pro mobilní aplikace je tento výraz nepřesný. Důvodem je, že u nasazování webové aplikace je nahrána nová verze softwaru na server a od té chvíle ji všichni uživatelé používají. U mobilních aplikací je nová verze nahrána na publikační server, ze kterého lze poté tuto verzi aplikace stáhnout a nainstalovat do zařízení. Po nahrání aplikace na server, již ale nikdo z vývojového týmu nemá vliv na to, kdy přesně si uživatel novou verzi nainstaluje, či tak neučiní. Nejpoužívanější publikační server pro systém Android je Google Play, který je defaultně předinstalován ve většině mobilních zařízení s tímto systémem. Pro iOS je to Appstore, který je vždy součástí zařízení.

Kapitola 2

Analýza služeb

Abychom mohli automatizovat procesy ve vývoji, testování a distribuci je nutné zvolit vhodné služby a programy, které automatizaci podporují. Kritéria pro výběr jsem nastavil tak, aby byly co nejvýhodnější pro firmu o padesáti zaměstnancích, která spravuje okolo dvaceti pěti aplikací. Prvním kritériem je cena služeb, jelikož automatizace by měla snížit potřebu zdrojů a tím i náklady firmy. Dalším důležitým kritériem je možnost správy uživatelů, tedy jak jejich přidávání a odstraňování, tak i nastavování přístupových práv. Zároveň vybrané služby a programy musí být mezi sebou kompatibilní, aby jejich automatizace byla jednoduchá a vůbec proveditelná.

2.1 Repozitář

Repozitář je vzdálené úložiště, které slouží pro uchovávání souborů. Umožňuje vytváření a editování více verzí zdrojových kódů a dalších souborů. Dnešní repozitáře již neslouží pouze k ukládání souborů, ale ke spojování různých verzí zdrojového kódu, správě úkolů, správě uživatelů a jejich práv. Níže porovnávané repozitářové služby tyto funkce mají a zároveň podporují systém pro správu verzí Git, který je hojně využívaný v softwarových projektech. Pro automatizaci procesů je nezbytné, aby služba rozeznávala, že do jednoho z repozitářů byly nahrány nové zdrojové kódy a tuto informaci uměla předat automatizačním nástrojům.

2.1.1 Github

Služba Github obsahuje nevíce repozitářů na světě. Je zde uloženo přes 67 milionů projektů [11]. Velkým problémem je, že Github v neplacené verzi neumožňuje zakládání privátních repozitářů. To samozřejmě není možné u projektů na zakázku, nebo u vlastnách produktů, aby jejich zdrojové kódy byly volně dostupné. Proto služba nabízí takzvané plány, které za měsíční poplatek umožňují využívat nadstandardní funkce. Nejlevnější plán, který podpo-

ruje vytváření privátních projektů a správu uživatelů organizace, se nazývá „Team“. Cena plánu je 25 $ měsíčně s tím, že je v organizaci pět uživatelů [12]. Každý další uživatel zvyšuje cenu o 9 $. Pro firmu o padesáti zaměstnancích tedy plán vychází na 430 $ měsíčně. Další plán se nazývá „Bussines“, za který je účtováno 21 $ měsíčně za uživatele a je nabízen ve dvou variantách. První varianta nabízí 24 hodinovou podporu v pracovní dny a zajištění 99.95 % dostupnosti [12]. Druhá varianta umožňuje provoz služby na vlastním stroji. Tento plán tedy pro padesát uživatelů vychází na 1050 $ měsíčně.

2.1.2 Bitbucket

Službu Bitbucket vyvíjí firma Atlassian, která zároveň vytváří světoznámé produkty jako Jira, nebo Confluence. Díky tomu jsou tyto systémy spolu velmi dobře provázány. Bitbucket umožňuje pro 5členný tým zakládat privátní projekty zcela zdarma. Větší týmy již musí platit 2 $ za každého člena [13]. Proto pro 50člennou firmu vyjde plán „Standard“ na 100 $ měsíčně. Bitbucket taktéž nabízí dražší plán, který umožňuje pokročilejší nastavení v zabezpečení a administraci. Ten je zpoplatněn 5 $ za uživatele měsíčně. Při zvolení tohoto plánu by firma platila 250 $ měsíčně.

2.1.3 Gitlab

Gitlab je open-source projekt a lidé z celého světa tedy mohou tuto službu rozvíjet a vylepšovat dle potřeby. Gitlab nabízí celkem osm plánů [14]. Z toho čtyři jsou pro variantu, kdy je Gitlab provozován na vlastním stroji. Zbylé čtyři plány jsou pro službu, která je provozována na oficiálních stránkách. Plány „Core“ a „Free“ jsou zcela zdarma a nabízí stejné funkce. Jediná odlišnost mezi nimi je pro jakou variantu služby jsou určeny. Již tyto základní plány nabízí neomezený počet privátních repozitářů s neomezeným počtem uživatelů. U varianty s vlastním provozem služby firma navíc získává administrátorská práva. Ta lze například využít pro nastavení přihlášení, nebo připojení Gitlab Runneru, který je více rozveden v sekci 2.2.2. Další plány nabízí oficiální podporu a přidávají funkce pro řízení projektů, jako například lepší správu úkolů a kontrolu nad repozitářem [15].

2.2 Automatizační nástroje

Po výběru repozitáře přichází na řadu výběr služby, která bude provádět automatizaci procesů. Repozitář předá informaci do jaké větve, nebo pod jakým tagem byl nahrán nový zdrojový kód. Na základě této informace služba pustí sekvenci předem nastavených úloh. Tyto úlohy jsou definovány v souboru, který je součástí zdrojových kódů. Při výběru automatizační služby je nutné zohlednit několik aspektů.

První z nich je operační systém, na kterém budou úlohy prováděny. Sestavení aplikace pomocí příkazů, které fungují na systému Linux, nemusí být kompatibilní s Windows. Dalším aspektem je, že služby často omezují celkovou dobu běhu automatizovaných úloh. Toto číslo se udává v minutách a pohybuje se v řádu stovek a každý měsíc se obnovuje. Posledním důležitým aspektem je počet paralelních úloh v jeden okamžik, které služba umožňuje provádět. Příkladem může být to, kdy je nahráno více zdrojových kódů do repozitářů ve velmi krátkém odstupu. Pokud služba umožňuje provádění pouze jedné automatizované úlohy současně, budou úlohy pro další repozitáře vyčkávat do doby, než bude fronta úloh z prvního repozitáře kompletně dokončena. Vždy je tedy nutné nastavit správný počet paralelních běhů tak, aby to vývojové týmy neomezovalo při práci a zároveň jich nebylo zbytečně mnoho.

2.2.1 Circle CI

Circle CI nabízí operační systémy Linux a macOS. Služba podporuje pouze dvě repozitářové služby, a to Bitbucket a Github. S Gitlab repozitářem tedy není kompatibilní a nelze procesy automatizovat [16].

Pro Linuxový systém je cena nastavena 50 $ za jeden stroj s tím, že první stroj je zdarma [17]. Pokud je objednaný pouze jeden stroj, a proto tedy není nutné nic platit je nastaven limit běhu úloh na 1500 minut měsíčně. Při koupi minimálně dvou strojů již časové omezení neplatí.

Pro macOS Circle CI nabízí čtyři plány [18]. Nejlevnější z nich nabízí dva stroje s omezením 500 minut automatizovaných úloh měsíčně. Tento plán stojí 39 $ měsíčně a je omezen maximálním počtem uživatelů na dva. O třídu vyšší plán stojí 129 $ měsíčně a nabízí 5 stojů s 1800 minutami pro běh automatizovaných úloh. Počet uživatelů již u tohoto plánu není omezen. Další plány již pouze navyšují počet strojů a počet minut.

2.2.2 Gitlab CI

Gitlab CI je již plně integrován ve službě Gitlab. Tento automatizační nástroj nelze využít pro služby Bitbucket a Github. Gitlab CI nenabízí stroje pro běh automatizovaných úloh, ale nabízí software s názvem Gitlab Runner. Runner po instalaci na stroj a prvotním spojením se službou Gitlab již může provádět automatizované úlohy. Runner stejně jako Gitlab je open-source, a je tedy zcela zdarma. Lze jej instalovat na všechny známe operační systémy, jako Windows, Linux a macOS. Jediné za co může být účtováno, jsou zakoupené stroje. Maximální doba běhu automatizovaných úloh není nijak omezena. Počet paralelních úloh záleží na počtu instalovaných runnerů. V tomto směru je velmi jednoduché škálování. Toto řešení platí pouze pro Gitlab provozovaný na vlastním stroji, nebo v placeném plánu. Pro možnost spojení Gitlab služby s runnerem je nutné mít administrátorská práva. U neplaceného plánu

lze zdarma využít sdílené runnery připravené od služby Gitlab, ale je zde omezeni 1800 minut měsíčně.

2.2.3 Jenkins

Jenkins je z výše uvedených automatizačních nástrojů nejdéle na trhu. První verze tohoto nástroje byla publikována již v roce 2011 [19]. Jenkins stejně jako Gitlab CI nenabízí připravené stroje, ale software pro správu automatizovaných úloh a programy, které tyto úlohy zpracovávají. Tyto programy se nazývají Agenti a lze je instalovat na Windows, Linux a macOS. Stažení a instalace je zcela zdarma, protože se jedná o open-source projekt. Na rozdíl od Circle CI a GItlab CI, tak automatizované úlohy nejsou definovány v souboru se zdrojovými kódy, ale definují se v samotné službě. Proto v roce 2017 Jenkins představil nový způsob automatizování úloh nazvaný Blue Ocean [20]. Ten již pracuje na stejném principu, jako předešlé služby.

2.3 Publikační servery

Pro distribuci mobilních aplikací existují oficiální publikační servery, kde lze publikovat i verze aplikací, které ještě nejsou určeny pro koncové uživatele. Tyto servery jsou specifické pro každý operační systém, a nelze tedy spravovat aplikace na jediném publikačním serveru zároveň. Dalším problémem je, že lze publikovat pouze jednu verzi aplikace současně. Toto může být problém v případě, když aplikace má více variant a je nutné otestovat všechny její varianty. Oficiální servery taktéž posuzují, zda nahrané aplikace nejsou škodlivé pro uživatele, nebo zařízení. To je samozřejmě důležité, ale posuzování může trvat i několik hodin. Což pro interní testování není z časových důvodů výhodné.

2.3.1 HockeyApp

HockeyApp je publikační server pro distribuci aplikací, který podporuje nejen operační systémy Android a iOS, ale i macOS a Windows. Služba rovněž nabízí zachytávání chyb a pádů aplikací. Ty jsou přiřazovány k distribuovaným verzím na serveru a tak lze jednoduše analyzovat kvalitu dané aplikace. Po nahrání aplikace neprochází žádnou kontrolou a je tedy ihned ke stažení. Od 1. ledna 2018 lze publikovat neomezený počet aplikací pro neomezený počet uživatelů zcela zdarma [21].

2.3.2 AppBlade

Služba AppBlade nabízí stejné funkce jako HockeyApp. Lze zde publikovat více verzí stejné aplikace. Taktéž při implementaci knihovny do zdrojových kódů služba zachytává chyby a pády aplikace. Velkým rozdílem je, že tato

služba je placená. Účtování neprobíhá za registrovaného uživatele do služby ani za projekty v ní vytvořené, ale za registrované zařízení. Všechny nabízené plány mají prvních dvacet pět zařízení zdarma [22]. Nejlevnější plán s názvem „Indie“ nabízí možnost publikovat Android a iOS Ad-Hoc aplikace, který stojí 1 $ měsíčně za zařízení. V plánu o třídu vyšší již lze navíc publikovat iOS Enterprise aplikace. Rozdíl mezi Ad-Hoc a Enterprise iOS aplikací je ten, že Ad-Hoc vyžaduje unikátní klíče zařízení, na které bude aplikace instalována. iOS aplikace v Enterprise variantě již lze instalovat bez nutnosti znát číslo zařízení. Tento vyšší plán stojí 2 $ měsíčně za zařízení.

2.4 Služby pro statickou analýzu

V základu většina služeb, které jsou orientované na statickou analýzu, nabízí stejné funkce. Umožňují vyhledávání chyb a bezpečnostních rizik ve zdrojovém kódu. Zobrazují procentuální pokrytí kódu automatizovanými testy, vyhledávají duplicity v souborech a vytváří statistiky ze zdrojového kódu v průběhu vývoje. Taktéž pokrývají většinu hojně používaných programovacích jazyků.

Služba s názvem SonarQube je open-source projekt. Umožňuje bezplatnou instalaci na vlastní server, nebo zdarma využívání oficiálních stránek, ale pouze v případě, že jsou analyzované projekty open-source [23]. Možnost analyzování privátních projektů na oficiálních stráncách je zpoplatněno 10 $ měsíčně za 100 tisíc řádků analyzovaného zdrojového kódu [24].

Naproti tomu služba Codacy nabízí analýzu kódu pouze přes oficiální stránky. Pro open-source projekty je tato služba zdarma, ale umožňuje spojení pouze s Bitbucket a Github. Plán, který nabízí analýzu privátních projektů, stojí 15 $ měsíčně za uživatele. Počet řádků zdrojových kódů ani počet projektů není omezen. Pro možnost analyzovat projekty uložené ve službě Gitlab je nutné zakoupit „Enterprise“ řešení.

2.5 Výběr služeb

Repozitářové služby nabízí ve svém základu téměř stejné funkce. Velkou odlišností je vytváření privátních projektů. Dle tabulky 2.1 vychází nejlépe cenově služba Gitlab. Ta je nabízena ve dvou variantách. Při instalaci služby v bezplatném plánu navíc uživatel získá administrátorská práva. Díky nim lze lépe nastavit pravidla v repozitářích a jednodušeji spravovat uživatele. Poté je ale potřeba stroj, na kterém bude služba provozována, zakoupit a udržovat. Tato varianta taktéž umožňuje vlastní správu strojů, na kterých budou prováděny automatizované úlohy pomocí programu Gitlab Runner. Tím odpadá nutnost zakoupit či instalovat službu pro automatizování procesů a v ní provádět další správu uživatelů.

Pro distribuci aplikací je výhodnější služba HockeyApp, jelikož je od roku 2018 zdarma. Oproti AppBlade neomezuje distribuci různých verzí iOS apli-

Služba	Cena	Počet uživatelů	Počet projektů
Github Team	430 $	50	neomezeně
Github Bussines	1050 $	50	neomezeně
Gitlab	0 $	neomezeně	neomezeně
Gitlab self-hosted	0 $	neomezeně	neomezeně
Bitbucket Standard	100 $	50	neomezeně
Bitbucket Premium	250 $	50	neomezeně

Tabulka 2.1: Srovnání repozitářových služeb

kace a ani neomezuje počet zařízení. O statickou analýzu se bude starat služba SonarQube. Ta může být instalována na stejný stroj jako Gitlab.

Finální cena se tedy počítá pouze ze zakoupených strojů. Požadavky na zdroje jednotlivých služeb jsou popsány v sekci 4.1. Náklady na stroje pro modelovou firmu jsou vyčísleny v sekci 5.2.

Kapitola 3

Pracovní postup

Automatizace procesů převádí část práce z člověka na stroj. Zatím nelze přenést veškerou práci, a je tedy nutné nastavit vstupy a výstupy procesů, které se provádí mezi člověkem a strojem. Takto standardizované procesy zajistí, že výstupy budou vždy dle očekávání. Pokud procesy nebudou dodržovány dle pravidel, může dojít k tomu, že stroj nebude schopen daný proces zpracovat a dojde k nechtěnému výstupu.

3.1 Před automatizací

Předtím, než byly zavedeny automatizační nástroje, bylo sestavování aplikace prováděno na stroji vývojáře. Sestavování aplikace bylo prováděno přes IDE, které si mnohdy uchovává dodatečné soubory pro rychlejší sestavení aplikace. To mohlo vést k tomu, že při změně části zdrojového kódu se aplikace mohla sestavit s kódem starým, nebo kombinací starého a nového kódu.

Dalším problémem byla distribuce. Aplikace byly nahrávány na publikační službu ručně, nebo byly distribuovány dalším testerům přes komunikační kanály umožnující posílání souborů. To samozřejmě mohlo vést k problému, kdy lidskou chybou byla distribuována chybná verze aplikace. To mohlo být zapříčiněno jak sestavením špatné varianty aplikace, tak i nahráním jiné varianty na publikační server.

Automatické testy byly taktéž prováděny pouze na stroji vývojáře, kde se objevuje stejný problém jako při sestavování aplikace. Výsledky testů tedy nebyly automaticky předávány dalším členům týmu, jelikož byly prováděny lokálně.

3.2 Git workflow

Git workflow jsou pravidla v systému pro správu verzí, která by měl vývojář dodržovat. Tato pravidla nastavují jmennou konvenci pro vývojové větve

a tagy v projektu. Taktéž popisují nutné podmínky pro zdrojový kód v jednotlivých větvích. Automatizované procesy mohou být na tato pravidla nastaveny, a tím bude vždy zajištěno provádění stejných a očekávaných automatizovaných úloh. Mnou sestavená Git workflow čerpá z pravidel Gitflow, která byla představena v roce 2010 [25].

Hlavní vývojovou větví je „devel“. V této větvi musí být zdrojový kód vždy sestavitelný a spustitelný. Zároveň musí procházet všechny testy. Vývoj nových funkcí bude probíhat ve větvích „feature/názevFunkce“. V těchto větvích není podmínkou sestavitelnost a plná úspěšnost testů. Při slučování jedné z větví „feature/..“ do větve „devel“ jsou pravidla nastavena tak, že všechny testy musí procházet úspěšně a aplikaci lze sestavit a spustit.

Když je aplikace připravena k většímu testovaní, je z větve „devel“ nahrán zdrojový kód do větvě „uat“. V této větvi neprobíhá aktivní vývoj, ale pouze se opravují chyby z testování. Tato větev musí být tedy vždy sestavitelná a průchod testy musí být 100%.

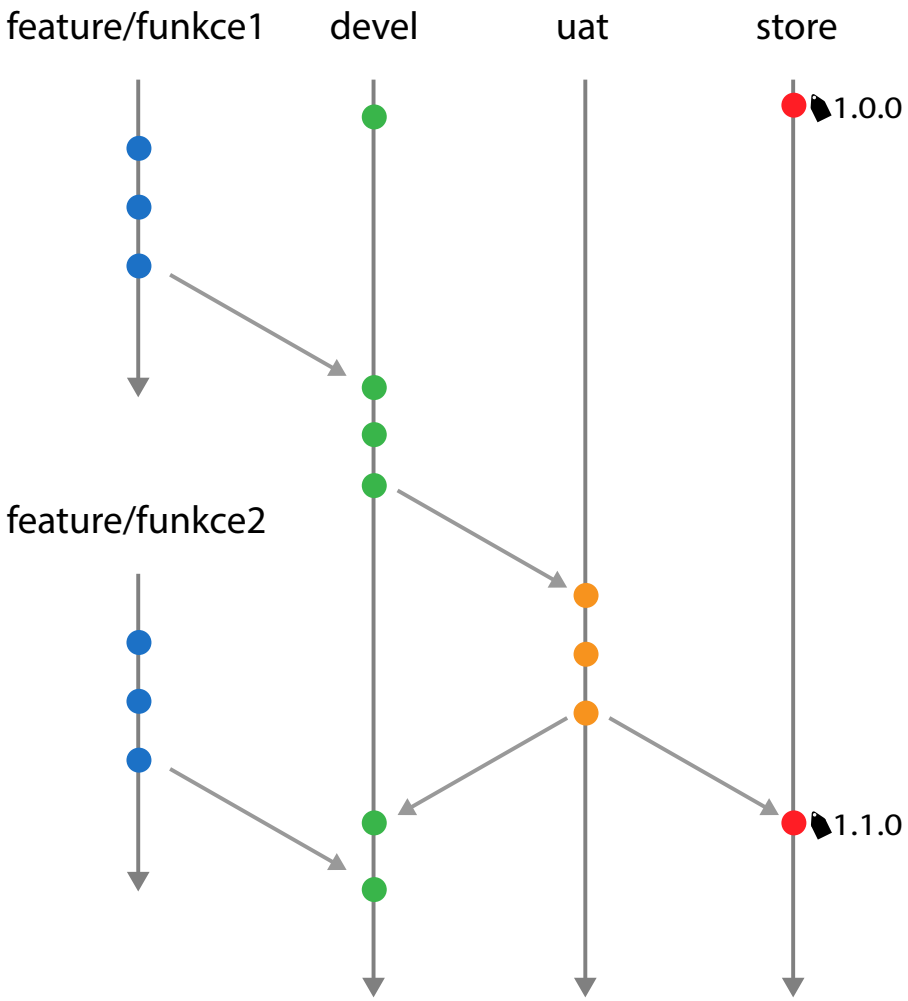
Pokud je aplikace ve větvi „uat“ bezchybná a zároveň je připravena na vydání pro koncové uživatele, je nahrána do větve „store“. Pokud ve větvi „uat“ byly prováděny opravy, je nutné tyto změny přenést i do větve „devel“, aby při dalším vydání aplikace nebylo nutné opravovat stejné chyby. Ve větvi „store“ je poté vytvořena značka s číslem verze aplikace. Celá Git workflow je znázorněna na obrázku 3.1.

3.3 Testování a distribuce aplikace

Služba HockeyApp umožňuje vytvářet až čtyři varianty stejné aplikace. Jmenovitě to jsou „Alpha“, „Beta“, „Enterprise“ a „Store“. Mezi variantami není žádný rozdíl. Výjimkou je „Store“ varianta, u které nelze nahrané aplikace stahovat z HockeyApp. Tato varianta je určena pouze k uchování aplikace a distribuce je prováděna přes oficiální publikační servery. Na základě stanovené Git workflow využiji pouze tři varianty.

Upravením zdrojových kódů ve větvi „devel“ bude výsledná aplikace nahrána do varianty „Alpha“. K této variantě má přístup pouze vývojový tým. Nahraná aplikace je ihned připravena k distribuci na testovací zařízení. Testovací tým může následně reportovat vývojovému týmu, zda úpravy ve zdrojovém kódu jsou bezchybné.

Když jsou zdrojové kódy přeneseny do větve „uat“, tak je prvně sestavená aplikace nahrána do „Alpha“ varianty ve službě HockeyApp. Pokud testovací tým označí tuto variantu za bezchybnou, je aplikace nahrána do „Beta“ varianty. K této variantě má přístup celý vývojový tým a zároveň větší okruh testerů. Větším okruhem testerů se u zakázkového vývoje myslí i zákazník. Za nahrání aplikace do „Beta“ varianty je odpovědný jeden ze členů testovacího týmu. Ten se musí přesvědčit, zda aplikace byla dostatečně testována a je bezchybná.



Obrázek 3.1: Git Workflow

Při vytvoření tagu je proces stejný jako při nahrání zdrojových kódů do větve „uat“. Navíc je aplikace po testovaní většího okruhu testerů nahrána do „Store“ varianty na HockeyApp a na oficiální publikační servery. Za nahrání aplikace na „Store“ variantu je zodpovědný projektový manažer. U zakázkového vývoje je nutné pro nahrání získat povolení od zákazníka. Sekvenci jednotlivých procesů popisuje obrázek 3.2. V obrázku je zohledněn pouze ideální průchod všemi procesy. Pokud jeden z procesů v průběhu selže je nutné problém odstranit a znovu projít celou sekvenci procesů.

Obrázek 3.2: Průchod procesy na základě Git Workflow

Kapitola 4

Realizace řešení

Na základě vybraných služeb z kapitoly 2 lze nyní nainstalovat potřebný software. Pro konfiguraci služeb poslouží navržené pracovní postupy z kapitoly 3.

4.1 Příprava serveru

Vzhledem k tomu, že ve výběru služeb byly zvoleny služby a programy, které vyžadují instalaci na vlastní stroj, je nutné, aby tento stroj obsahoval dostatečný výpočetní výkon. Gitlab požadavky na zdroje škáluje dle počtu uživatelů využívajících službu. Pro padesát uživatelů služba doporučuje minimálně jedno jádro, 2GB RAM a 2GB SWAP paměti [26]. Pro optimální běh doporučuje dvě jádra a 4GB RAM. Ohledně paměti Gitlab neudává žádné doporučení, jelikož vše závisí na počtu repozitářů uchovaných ve službě a jejich velikosti.

Gitlab Runner nemá žádné požadavky na zdroje serveru. Jediným doporučením je neinstalovat runner na stejný stroj jako Gitlab. Zdůvodnění spočívá v tom, že když runner provádí automatizovanou úlohu, využívá tím i zdroje serveru, které se poté nedostávají pro službu Gitlab a tím ji zpomalují.

SonarQube doporučuje nejméně 2GB RAM pro službu a 1GB pro operační systém [27]. Požadavky na počet jader serveru a velikost paměti nezmiňuje. Udává modelovou instanci, která během čtyř let analyzovala přes 30 milionů řádků zdrojových kódů a obsahuje přes 800 projektů. Tato instance zabírá okolo 25GB paměti na disku s tím, že 15GB tvoří databáze služby a zbylých 10GB paměti tvoří samotná služba.

Pro realizaci řešení jsem zvolil server s parametry 2GB RAM, dvěma jádry a 50GB paměti s maximálním přenosem dat 5TB měsíčně. Na tomto serveru poběží všechny tři služby zároveň. Takovéto nastavení nesplňuje všechna doporučení, ale pro otestování řešení s jedním uživatelem je dostačující. Na serveru je předinstalována linuxová distribuce Ubuntu ve verzi 16.04. Instalační kroky níže popsané jsou tedy specifické pro tento systém.

4.2 Instalace programů

4.2.1 Gitlab

Gitlab doporučuje instalaci na linuxové operační systémy. Taktéž je podporován službami nabízejícími dedikované servery jako AWS a Microsoft Azure. Před samotnou instalací služby je nutné stáhnout několik balíčků, které jsou nutné pro instalaci a běh služby Gitlab. Jedná se o balíčky „curl“, „openssh-server“ a „ca-certificates“ [28]. Dalším krokem je stažení instalačního scriptu. Při stažení scriptu je nutné dbát na to, aby URL obsahovala správnou variantu služby. V URL se nachází buď „gitlab-ce“ nebo „gitlab-ee“. Ve variantě s koncovkou „ce“ je služba bezplatná. U varianty „ee“ se jedné o „Enterprise“ verzi, která je placená. Spuštěním tohoto scriptu se nastaví do systému potřebné klíče a odkazy na stažení správné verze.

Před instalací je nutné nastavit proměnnou prostředí „EXTERNAL_URL“ na které po instalaci bude služba dostupná. Následně již stačí spustit instalaci pomocí „apt-get install gitlab-ce“. Pro zjištění stavu, zda je služba spuštěna, slouží příkaz „gitlab-ctl status“. Výpis stavu služby, je možné vidět níže. Pokud služba není v provozu, je možné ji zapnout pomocí příkazu „gitlab-ctl start“.

```
sudo gitlab-ctl status
run: gitaly: (pid 40507) 68s; run: log: (pid 26159) 191s
run: gitlab-monitor: (pid 40519) 68s; run: log: (pid 26319) 191s
run: gitlab-workhorse: (pid 40522) 68s; run: log: (pid 26096) 191s
run: logrotate: (pid 4584) 243; run: log: (pid 26137) 191s
....
```

4.2.2 Gitlab Runner

Instalace programu Gitlab Runner je obdobná jako u služby Gitlab. Pokud je runner instalován na jiný stroj než Gitlab, je nutné znovu stáhnout balíček „curl“ [29]. Následně stačí stáhnout instalační script. Po nastavení potřebných odkazů a klíčů pomocí scriptu již stačí spustit samotnou instalaci. Ta probíhá pomocí příkazu „apt-get install gitlab-runner“. Při potřebě instalace specifické verze je nutné přidat na konec příkazu rovnítko a číselnou verzi runneru. Následně je nutné runner ještě s Gitlab službou spárovat. Tento proces je popsán v sekci 4.3.2.

4.2.3 Android

Na stroji, kde budou probíhat automatizované úlohy, je navíc nutné nainstalovat i balíčky umožňující sestavení aplikace pro daný operační systém. U systému Android se jedná o Android SDK. Před instalací SDK je nutné na Ubuntu povolit architekturu i386. Dále je nutné nainstalovat balíčky „libc6:i386“,

„libstdc++6:i386“ a „libz1:i386“. Dalším krokem je stažení Android SDK pro linuxovou distribuci z Google repository [30]. SDK je po stažení uložené v archivu, a je tedy nutné mít na stroji nainstalovaný balíček pro „unzip“. Po rozbalení do složky je zapotřebí nastavit proměnnou systému „PATH“. Ta musí navíc obsahovat cestu do složek v SDK „platform-tools“ a „tools“. Každá Android aplikace obsahuje proměnné „compileSdkVersion“ a „targetSdkVersion“. Ty určují verzi Android systému, pro který je aplikace optimalizována a s jakou verzí SDK se má aplikace sestavovat. Tyto verze musí být nainstalovány na stroji, aby došlo k úspěšnému sestavení aplikace. Pomocí příkazu „sdkmanager“ lze tyto části SDK stahovat. Pro základní sestavení aplikace je nutné instalovat:

- platform-tools,
- platforms;android-27,
- build-tools;27.0.3,
- extras;android;m2repository,
- extras;google;m2repository,
- extras;google;google_play_services.

Pro běh UI testů je zároveň potřeba mít na stroji nainstalovaný a vytvořený emulátor. Pomocí programu „sdkmanager“ je zapotřebí stáhnout balíček „emulator“ a obraz systému Android. Tyto systémy jsou vydávané podle verzí operačního systému Android a podle architektury. Pro architekturu x86 je zapotřebí, aby operační systém na stroji podporoval KVM. Příkaz „avdmanager create avd“ s přiloženým názvem emulátoru a názvem staženého obrazu systému vytvoří emulátor. Tento emulátor lze spustit příkazem „emulator -avd nazev“. Pro rychlejší nastartování emulátoru na serveru je dobré přidat k příkazu parametry „-no-audio“ a „-no-window“.

Pro plný výčet SDK balíčků lze použít příkaz „sdkmanager –list“. Nakonec je nutné schválit licence a toho lze docílit pomocí „sdkmanager –licenses“.

4.2.4 SonarQube

SonarQube pro běh vyžaduje nainstalovanou databázi. Podporuje Microsoft SQL server, PostreSQL a Oracle [31]. Na systému Ubuntu je možné jedním příkazem nainstalovat PostreSQL. Po spuštění PostreSQL je nutné založit databázi a uživatele, který bude přiřazen jako vlastník dané databáze.

Po stažení a rozbalení SonarQube je zapotřebí nastavit přístupové údaje do databáze. Ty se nastavují v souboru, který se nachází na cestě „conf/sonar.properties“. Defaultně nastavený SonarQube již obsahuje administrátorského uživatele pod přihlašovacím jménem „admin“ a heslem „admin“.

4.3 Nastavení automatizace

Veškeré potřebné programy a služby jsou nainstalovány. Stále ale nejsou správně nakonfigurovány tak, aby si informace mezi sebou správně předaly. Část služeb je potřeba nastavit pouze poprvé, jiné zase vyžadují nastavení vždy pro nový projekt.

4.3.1 Automatizační soubor

Automatizační soubor musí být uložen v kořenu repozitáře a nést název „.gitlab-ci.yml“. Soubor je psán v jazyce YAML, což je běžný jazyk pro konfigurační soubory. Na začátku souboru je možné nastavit proměnné, které jsou poté dostupné v jednotlivých automatizovaných úlohách. Taktéž je možné pomocí příznaku „before_script“ natavit příkazy, které proběhnou vždy před spuštěním automatizační úlohy. Pro sestavení Android aplikace se nejčastěji využívá nástroj Gradle. V kořenové složce projektu se poté nachází soubor „gradlew“. Aby bylo možné sestavení aplikace spustit, je nutné tomuto souboru dát práva na spuštění. To může být provedeno právě v „before_script“.

Dále lze v souboru definovat jednotlivé fáze pomocí příznaku „stages“. Do těchto fází se následně přiřadí jednotlivé automatizované úlohy. Jak jsou jednotlivé fáze sepsány shora dolů, tak i v tomto pořadí budou automatizované úlohy probíhat. Je tedy nutné mít fázi sestavení aplikace nad fází, kde se aplikace nahrává na publikační server. Každá fáze může mít několik úloh najednou. Toho lze využít u jednotkového a UI testovaní, kde každé má vlastní úlohu, ale obě spadají do jedné fáze.

V úryvku níže je možné vidět nastavení základních proměnných potřebných pro analýzu a nahrání aplikace na HockeyApp. Taktéž jsou zde vytvořeny fáze ve správném pořadí.

```
variables:
  HOCKEY_APP_URL: "https://rink.hockeyapp.net/api/2/apps"
  HOCKEY_APP_APP_ID_ALPHA: "id-token"
  SONAR_URL: "http://sonar.tomaslala.cz/"

before_script:
  - chmod +x ./gradlew

stages:
  - build
  - analyze
  - test
  - deploy
```

Dále se v dokumentu nachází definice jednotlivých automatizovaných úloh.

Každá úloha začíná názvem. Dále obsahuje mnoho upřesňujících příznaků. Prvním z nich je „stage“, který definuje, do jaké fáze tato úloha přísluší. Dalším příznakem je „tags“, ten určuje, na jakém runneru úloha bude vykonána. Nastavení příslušné značky pro runner je popsáno v sekci 4.3.2. Gitlab CI taktéž podporuje výběr větví a značek, pro které se dané úlohy spustí, či nespustí. K tomu slouží příznaky „only“ a „except“. Příznak „only“ obsahuje názvy, pro které daná úloha bude spuštěna. Naproti tomu příznak „except“ řídí, pro jaké názvy úloha spuštěna nebude. Pro definici názvu je možné využít i regulární výrazy.

Jediný příznak, který je vždy pro definici úlohy vyžadován, je „script“. Ten obsahuje příkazy, které se provedou na příkazové řádce na stroji, kde je runner instalován. Posledním důležitým příznakem je „artifacts“. V tomto příznaku lze definovat cestu a dobu trvanlivosti. Při použití příznaku mají následující úlohy přístup k souborům na definované cestě. Toho se využívá tak, že v jedné úloze je aplikace sestavena a v další je nahrána na publikační server.

Ve zdrojovém kódu níže je popsána úloha pro sestavení aplikace. Úloha se nachází ve fázi „build“. Příkazy budou probíhat na stroji, který je na sestavení aplikace připraven. Toto je zajištěno pomocí značky „android“. Tato úloha se spouští pouze v případě větve „devel“, „uat“ a číselné značky. Sestavená aplikace je poté uchována po jeden den.

```
Build App:
  stage: build
  tags:
   - android
  only:
   - devel
   - uat
   - /^(0|[1-9][0-9]*)\.(0|[1-9][0-9]*)(\.(0|[1-9][0-9]*))?$/
  script:
   - ./gradlew assembleDebug
  artifacts:
    paths:
      - app/build/outputs/
    expire_in: 1 day
```

Pro testování jsem vytvořil dvě oddělené úlohy. Ty jsou rozděleny na jednotkové a UI testy tak, aby podle názvu bylo jasné jaký druh testů prochází, či nikoliv. Příznaky pro testy od příznaků pro sestavení se příliš neliší. Úlohy jsou zařazeny do testovací fáze. U jednotkových testů se aplikace nesestavuje, a není tedy třeba nic uchovávat. Pro UI testy je aplikace třeba, ale daný příkaz si aplikaci sestaví. Následující úryvek obsahuje pouze změněné části oproti úloze pro sestavení.

```
Unit tests:
  stage: test
  ...
  script:
   - ./gradlew testDebugUnitTest

UI tests:
  stage: test
  ...
  script:
   - ./gradlew connectedAndroidTest
```

Pro analýzu pomocí SonarQube lze využít oficiální plugin pro sestavovací nástroj Gradle. Po přidání pluginu začne soubor „gradlew“ nabízet nový příkaz s názvem „sonarqube“. Tento příkaz přijímá mnoho parametrů. Nejdůležitější z nich jsou pro nastavení cesty ke zdrojovým a testovacím kódům, nastavení URL SonarQube a autorizační údaje. Dobrým parametrem je „sonar.branch“, který rozděluje aplikaci do více projektů ve službě SonarQube. Díky tomu se nemíchají analýzy zdrojových kódů z vývojových větví se stabilními větvemi. V části níže, je příkaz „sonarqube“ spouštěn pouze s URL, větví a autorizačním tokenem. Zbylé informace jsou nastaveny ve zdrojových kódech aplikace. Proměnnou „CI_COMMIT_REF_NAME“ vytváří Gitlab sám a obsahuje název větve, nebo název značky. Autorizační token je uložen v tajných proměnných služby Gitlab, které jsou popsány v sekci 4.3.4.

```
Static analyze:
  stage: analyze
  ...
  script:
    - ./gradlew sonarqube -Dsonar.host.url=${SONAR_URL}
     ↪  -Dsonar.branch=${CI_COMMIT_REF_NAME}
     ↪  -Dsonar.login=${SONAR_QUBE_TOKEN}
```

Aplikace lze nahrát na službu HockeyApp pomocí vystaveného API [32]. Pro nahrání aplikace je nutné sestavit správnou URL, která pro identifikaci vytvořeného projektu na HockeyApp obsahuje identifikační číslo projektu. Úryvek níže obsahuje skládání správné URL. Takto definovaná proměnná je dostupná pouze pro danou úlohu.

Ještě před nahráním aplikace je nutné znát její umístění ve složkách. Při sestavení aplikace je instalační balíček pojmenován na základě vybrané varianty. Pokud automatizační soubor obsahuje více úloh, které sestavují různé varianty aplikace, nelze jasně určit její název v nahrávací úloze. Proto se před nahráním aplikace spustí příkaz, který ve složkách aplikaci najde. Vyhledávání

probíhá na základě stanovené cesty, která se při sestavení aplikace nikdy nemění, a koncovky „.apk“. Tato koncovka označuje instalační balíček Android aplikace.

Po nalezení aplikace proběhne její nahrání na HockeyApp pomocí balíčku curl. V příkazu je nutné nastavit „status=2“, který určuje, že aplikaci poté bude možné ze serveru stáhnout. V parametru „ipa“ je nutné nastavit cestu k instalačnímu balíčku. Pro autorizaci je přiložen token do hlavičky. Jeho získání je popsáno v sekci 4.3.3. Celý příkaz ukončuje sestavená URL, na kterou má být aplikace nahrána.

```
Alpha deploy:
  stage: deploy
  ...
  variables:
    UPLOAD_URL:
     ↪ "${HOCKEY_APP_URL}/${HOCKEY_APP_APP_ID_ALPHA}/app_versions/upload"
  script:
   - export version=$(find app/build/outputs/apk/ -name "*.apk"
    ↪ | head -1) # find build apk
   - 'curl -F "status=2" -F "ipa=@${version}" -H
    ↪ "X-HockeyAppToken: ${HOCKEY_APP_TOKEN}" ${UPLOAD_URL}'
```

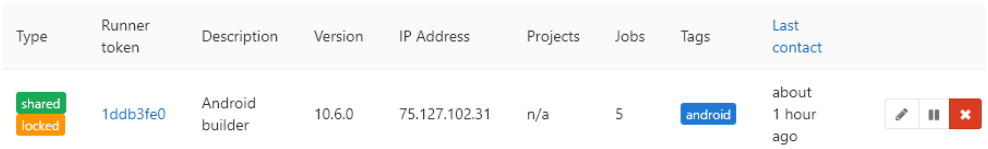
4.3.2 Gitlab a Gitlab Runner

Podmínkou, aby Gitlab služba posílala příkazy programu Gitlab Runner, je potřeba runner u služby zaregistrovat. Před začátkem registrace je dobré zjistit URL a autorizační token služby Gitlab. Tyto informace lze nalézt v administrátorské sekci pod položkou „Runners“. Následně již stačí zadat příkaz „sudo gitlab-runner register“ [33]. Tímto příkazem se spustí průvodce, který jako první bude požadovat URL, a následně autorizační token. Dále registrace vyžaduje popis runneru a značky. Tyto značky využívá automatizační soubor u jednotlivých úloh. Pomocí těchto značek lze definovat, jaké úlohy budou prováděny na jakých runnerech.

Následně runner vyžaduje specifikování, zda může provádět úlohy, které nemají nastavenou značku. Poté se táže, zda si má uzamknout projekty, na kterých provádí automatizované úlohy. Úlohy z tohoto projektu již nedostane žádný jiný runner. Poslední otázka specifikuje, kde budou příkazy probíhat. Pokud je správně runner nastaven, zobrazí se v administrátorské sekci pod položkou „Runners“. Na obrázku 4.1 je možné vidět registrovaný runner ve službě Gitlab.

4.3.3 Gitlab a HockeyApp

Nejprve je nutné vytvořit projekty ve službě HockeyApp. Při vytváření aplikace vyžaduje HockeyApp zadat operační systém, na který bude aplikace in-



Obrázek 4.1: Registrovaný Gitlab Runner

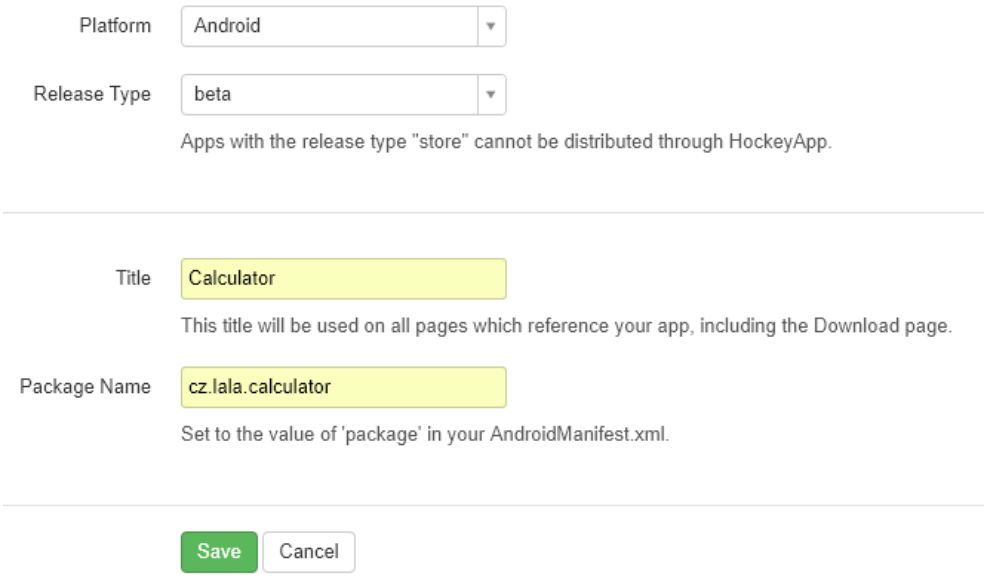
stalována. Následně je nutné zadat variantu aplikace, název aplikace a název balíčku aplikace. Název balíčku je jedinečný identifikátor aplikace. Na obrázku 4.2 je možné vidět vyplněné údaje pro testovací aplikaci. Po vytvoření aplikace je možné najít v přehledu identifikační číslo projektu. Toto číslo je nutné uložit do automatizačního souboru.

Vytvoření autorizačního tokenu je podmínkou pro to, aby runner mohl nahrávat aplikaci na HockeyApp. Token se vytváří v nastavení účtu uživatele. Uživatel tedy musí mít přístup k projektům, do kterých chce nahrávat aplikace přes token. Zároveň v projektu musí mít nastavená práva na nahrávání aplikací. Při vytváření tokenu služba vyžaduje specifikování, pro jaké projekty je možné token použít. HockeyApp nabízí možnost nespecifikování projektu a tím umožňuje token využívat na všechny přístupné projekty. Jako poslední se nastavují práva tokenu. Ty HockeyApp rozděluje na čtyři varianty. Umožňuje nastavení práv pro prohlížení, nahrávání aplikací bez možnosti distribuce, nahrávání aplikací s možností distribuce a poté neomezená práva.

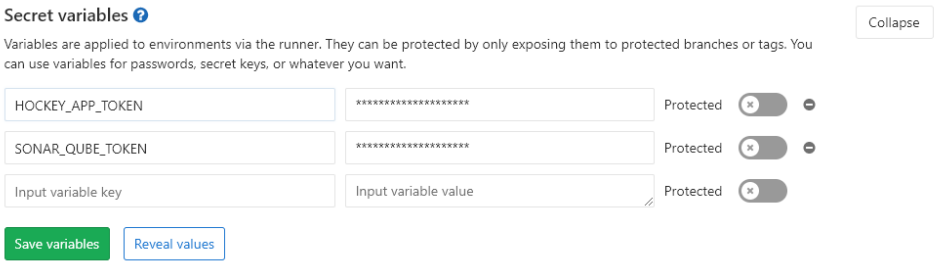
Pro Gitlab jsem zvolil práva nahrání a distribuce, aby aplikaci bylo možné ihned testovat. Token je uložen v tajných proměnných služby Gitlab. K tomuto nastavení nemají přístup testeři ani vývojáři, ale pouze majitelé projektu. Tato proměnná je následně nastavená do operačního systému při spuštění automatizované úlohy, tak jako proměnné z automatizačního souboru.

4.3.4 Gitlab a SonarQube

Pro nastavení SonarQube je nutné pouze vytvořit autorizační token. Projekt ve službě SonarQube se založí při nahrání první analýzy. Token lze vytvořit v nastavení účtu pod záložkou Zabezpečení. Zde stačí pouze zadat název tokenu, který je poté vygenerován. Token po vygenerování je nutné ihned uložit, jelikož při opakovaném načtení stránky již nelze zobrazit. Jediná možnost je token smazat a vytvořit nový. Stejně jako u HockeyApp je tento token uchován v tajných proměnných v projektu aplikace ve službě Gitlab. Výčet tajných proměnných následně vypadá jako na obrázku 4.3.



Obrázek 4.2: Formulář pro vytvoření HockeyApp projektu



Obrázek 4.3: Tajné proměnné služby Gitlab

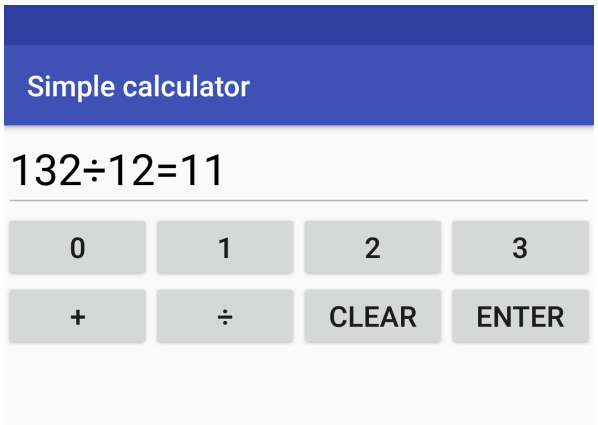
4.4 Vývoj mobilní aplikace

Pro test, zda automatizace správně sestaví mobilní aplikaci a provede příslušné testy, jsem vytvořil jednoduchou Android aplikaci. Její „targetSdkVersion“ je nastaveno na verzi 27 a taktéž i „compileSdkVersion“. Tyto verze jsou nainstalovány i na stroji s Android SDK, které zmiňuji v sekci 4.2.3.

4.4.1 Kalkulačka

Aplikace funguje jako jednoduchá kalkulačka. Nabízí operace sčítání a dělení. Pro zadávání čísel slouží tlačítka od 0 do 3. Následně aplikace obsahuje tlačítka na zobrazení výsledku a na vymazání čísel. Rozložení prvků je možné vidět na obrázku 4.4. Třída `Calculator` obsahuje pouze výpočetní logiku a není nijak závislá na zbylém zdrojovém kódu Android aplikace. Díky tomu, je možné ji testovat bez nutnosti simulovat či spouštět sestavenou aplikaci.

```
public class Calculator {

    public static int plus(int first, int second){
        return first + second;
    }

    public static int divide(int first, int second){
        if (second==0){
            throw new ArithmeticException();
        }
        return first / second;
    }
}
```



Obrázek 4.4: Aplikace Simple calculator

4.4.2 Jednotkové testy aplikace

Jednotkové testy jsou napsány v jazyce Java. Dle standardu se jednotkové testy nachází v adresáři „app/src/test“, kde „app“ je název modulu. Pro napsání jednotkového testu je nutné využít JUnit [34] knihovnu, která obsahuje metody

pro porovnávání stavů. Pro vytvoření testu je zapotřebí vytvořit třídu v daném adresáři. Poté už stačí jen vytvořit funkci, ke které se přidá anotace `@Test`. Každá funkce takto anotovaná je brána jako jeden samostatný test. Pokud se v testech dané třídy opakují počáteční či koncové kroky, je možné využít anotací `@Before` a `@After`. Takto označené funkce proběhnou vždy před, nebo po každém testu. V úryvku zdrojového kódu níže lze nalézt jednotkový test pro třídu `Calculator`. Jeden test zjišťuje, zda metoda `plus` funguje korektně pro různé kombinace čísel. Druhá funkce testuje, zda při volání metody `divide` s druhým nulovým parametrem třída zareaguje aritmetickou chybou.

```
public class CalculationUnitTest {

    @Test
    public void isPlusCorrect () throws Exception {
        assertEquals(1,Calculator.plus(1,0));
        assertEquals(0,Calculator.plus(1,-1));
        assertEquals(15,Calculator.plus(10,5));
    }

    @Test(expected = ArithmeticException.class)
    public void isDivideExceptionCorrect () throws Exception {
        Calculator.divide(30,0);
    }

}
```

4.4.3 UI testy aplikace

Testy UI jsou taktéž napsány v jazyce Java. Liší se ale adresář těchto testů. Ten se nachází v „app/src/androidTest/“. Pro UI testování se využívá framework Expresso [35]. Tento framework umožňuje vyhledávání grafických prvků na obrazovce a jejich interakci s nimi pomocí zdrojového kódu. Pomocí anotace třídy `@RunWith`, ve které je zvolen `AndroidJUnit4`, je možné psát UI testy velmi podobně jako ty jednotkové. Další nutnou anotací třídy je `@LargeTest`, která navyšuje čas na běh testů. Pro určení na jaké obrazovce budou testy probíhat slouží proměnná třídy `ActivityTestRule`. Proměnnou je nutné anotovat pomocí `@Rule` a vložit do ní obrazovku ve formě `Activity`.

Následně už stačí vytvořit funkce anotované `@Test`, které budou reprezentovat jednotlivé testy. V testech se primárně využívá funkce `onView`, která pomocí dalších specifik vyhledá prvek na obrazovce. Prvek je možno vyhledávat podle textu, který obsahuje nebo podle jeho ID. Na tomto prvku lze poté provést kliknutí nebo úpravu textu. Zároveň lze na prvku kontrolovat, zda je pro uživatele viditelný či obsahuje nějaký text.

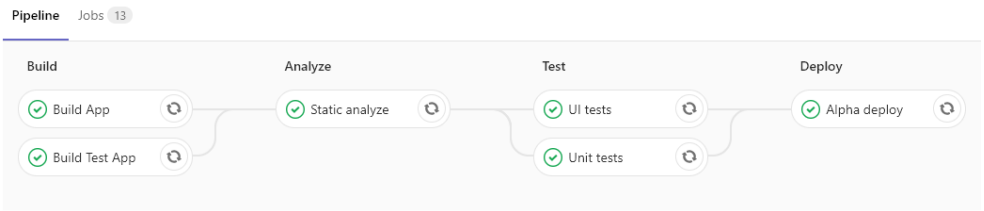
Tento druh testů tedy vyžaduje znalost obrazovky a její chování tak, aby test byl napsán správně. Při spuštění testu je nutné vybrat zařízení nebo emulátory, na kterých testy poběží. Třída níže testuje obrazovku `MainActivity`. Obsahuje dva testy, kde první z nich kontroluje, zda aplikace při dělení nulou ukazuje text, který tento problém popisuje. Druhý test zkouší validní dělení dvou čísel a zjišťuje, zda finální text odpovídá očekávanému výstupu.

```
@RunWith(AndroidJUnit4.class)
@LargeTest
public class ExampleInstrumentedTest {

    @Rule
    public ActivityTestRule<MainActivity> mActivityRule =
    new ActivityTestRule<>(MainActivity.class);

    @Test
    public void divideZeroAlert() throws Exception {
        onView(withId(R.id.btn_2)).perform(click());
        onView(withId(R.id.btn_divide)).perform(click());
        onView(withId(R.id.btn_0)).perform(click());
        onView(withId(R.id.btn_enter)).perform(click());
        onView(withText(R.string.txt_divide_zero)).check(matches(isDisplayed()));

    }

    @Test
    public void divideNumbers() throws Exception {
        onView(withId(R.id.btn_3)).perform(click());
        onView(withId(R.id.btn_divide)).perform(click());
        onView(withId(R.id.btn_1)).perform(click());
        onView(withId(R.id.btn_enter)).perform(click());
        onView(withId(R.id.edt_numbers)).check(matches(withText("3÷1=3")));
    }
}
```

4.5 Test procesů

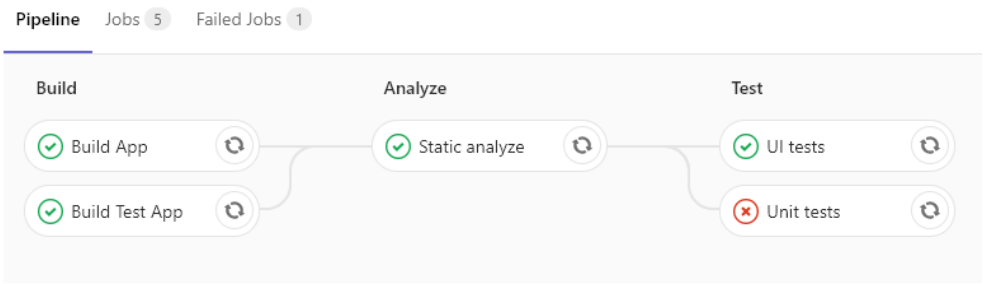
Nyní jsou služby a programy nastaveny. Zakoupený virtuální server nepodporuje KVM a nelze tedy na něm virtualizovat systém Android. Pro účely UI testů byl přidán nový runner, který je provozován na reálném stroji. Pro otestování tedy stačí nahrát zdrojové kódy do repozitáře. Po nahrání by se ve službě Gitlab v záložce CI/CD měla vytvořit sekvence úloh, která je definovaná v automatizačním souboru.

Po nahrání zdrojových kódů do větve „devel“ vznikla sekvence úloh zobrazená na obrázku 4.5. Všechny úlohy se dokončily úspěšně. Ve službě SonarQube vznikl projekt s názvem „Calculator devel“, který neobsahuje žádné závažné chyby. Na HockeyApp byla nahrána první verze aplikace.



Obrázek 4.5: Automatizované úlohy větve „devel“

Při nahrání kódů do větve „feature/dividefail“ byly vytvořeny úlohy, které je možné vidět na obrázku 4.6. Do zdrojových kódů byly zaneseny chyby a funkce `divide` byla upravena, aby vracela špatnou výjimku. SonarQube v projektu „Calculator feature/dividefail“ nalezl čtyři vážné chyby. Jednotkový test kvůli změně zdrojových kódů neprošel. Úloha na nahrávání aplikace se nespustila, protože nemá v automatizačním souboru definované „feature/“ větve.



Obrázek 4.6: Automatizované úlohy větve „feature/dividefail“

Služby, na kterých proběhlo testování procesů, jsou dostupné na stránce tomaslala.cz.

Kapitola 5

Přínosy a náklady

5.1 Přínosy

Největším přínosem automatizace je, že stroje vykonávají práci, kterou předtím musel vykonávat projektový tým. Vývojáři již nemusí sestavovat aplikace pro distribuční server na svých strojích. Členové týmu si nemusí předávat aplikace přes komunikační kanály či nahrávat ručně aplikace na distribuční server. Výstupy analýz a testů jsou uloženy na síti a jsou pro všechny členy dostupné.

Díky tomu, že jsou úlohy prováděny strojem, tak téměř zaniká možnost zanesení lidské chyby do procesu. Pro správné spuštění automatizovaných úloh vývojáři musí dodržovat nastavená pravidla pro správu verzí. Pokud pravidla nejsou dodržena, nedojde ke spuštění úloh.

Při správném pokrytí zdrojových kódů jednotkovými a UI testy nemůže dojít k zanesení nové chyby do aplikace. Testy se spouští ve všech fázích vývoje tak, aby došlo k co nejrychlejšímu odhalení problému. Statická analýza odhaluje možné chyby v kódu a standardizuje jeho zápis.

Díky standardizaci procesů je možný mnohem plynulejší přechod jednotlivých členů týmu z jednoho projektu na druhý, jelikož tyto procesy jsou uplatněny pro celou firmu. Ušetřený čas, který byl předtím věnován procesům prováděných manuálně, lze nyní využít pro zlepšení projektu. Všechny tyto přínosy následně zvyšují konkurenceschopnost na trhu.

5.2 Náklady

Všechny využité služby byly zvoleny v bezplatné verzi. Nákladem jsou tedy stroje, na kterých služby budou provozovány. Gitlab a SonarQube dle jejich doporučení by byly nainstalovány na server s 4GB RAM, 2 jádry a 100GB paměti. Dle tabulky 5.1 stroj v podobných parametrech vychází od 300 Kč do 484 Kč. Ceny jsou uváděny v měsíčních platbách včetně DPH.

Pro provoz runnerů nejsou stanovené minimální parametry. Při vykonávání úlohy runner primárně využívá zdroje procesoru a RAM paměti. Runner

Služba	Počet jader	Velikost RAM	Velikost paměti	Cena
axfone.cz	2	4GB	100GB	300 Kč
master.cz	2	4GB	100GB	469 Kč
wedos.com	2	6GB	90GB	484 Kč
forpsi.com	2	4GB	80GB	363 Kč

Tabulka 5.1: Srovnání serverů pro služby

tedy není nutné provozovat na stroji s velkou kapacitou pevných disků. Dle doporučení Gitlab by runner neměl být provozován na stejném stroji jako repozitářová služba. Proto je dobré pro každý runner zakoupit vlastní stroj. Počet strojů se odvijí od počtu aktivních projektů v Gitlab službě. Úlohy spojené s Android aplikacemi lze provádět na systémech Windows, macOS a Linux. Pro iOS aplikace je nutné využít macOS.

Dalším nákladem je osoba, která celé řešení zrealizuje. Osoba musí mít základní administrátorské znalosti. Po nasazení řešení je nutné aktualizovat a sledovat vytížení služeb. Na základě zatížení serverů následně bude rozšiřovat či redukovat zdroje pro služby.

Posledním nákladem může být školení projektového týmu na nové procesy. Nasazení nových procesů může projektový tým na určitý čas zpomalit z důvodu, že tým nemá tyto procesy plně osvojeny. Vše závisí na tom, jak razantně byly nové procesy změněny, případně, zda nějaké přibyly.

Závěr

Cílem této práce bylo provést analýzu dostupných služeb pro automatizaci procesů a z vybraných služeb realizovat řešení. Při analýze byl kladen důraz na finanční stránku služby a funkce umožňující spravovat uživatele. Finanční částky byly vypočítávány na základě modelové firmy.

Před samotnou realizací jsem nastavil pracovní postup pro projektový tým. Tento postup obsahuje pravidla pro správu verzí zdrojového kódu. Pro testery a projektové manažery definuje jejich odpovědnost. Postup taktéž stanovuje instrukce pro správnou distribuci aplikací. Tato standardizace procesů projektového týmu umožní nastavení služeb s jasnými pravidly. Výstupy služeb budou následně vždy konzistentní a očekávané.

Pro realizaci řešení jsem vybral služby Gitlab, Gitlab Runner, SonarQube a HockeyApp. Po instalaci služeb jsem provedl jejich konfiguraci, aby služby mezi sebou správně komunikovaly. Pro otestovaní, zda realizace celého řešení funguje, jsem vytvořil Android aplikaci s jednotkovými a UI testy. Po nahrání zdrojových kódů do repozitáře byla vytvořena sekvence automatizovaných úloh. Sekvence úloh přesně odpovídala nadefinovaným procesům v pracovním postupu.

Jako první proběhlo sestavení aplikace ve správné variantě. Následně zdrojové kódy byly podrobeny statické analýze. Po analýze byla aplikace otestována pomocí jednotkových a UI testů. Po úspěšném dokončení všech úloh byla aplikace v poslední fázi nahrána na distribuční server.

V budoucnosti je možné řešení vylepšit o nově vznikající služby, které usnadní či zkrátí vývojové procesy. Realizace řešení je zaměřena na systém Android, proto by bylo možné realizaci rozšířit o systém iOS a multiplatformní vývoj.

Literatura

[1] Mobile Operating System Market Share Worldwide [online]. [cit. 2018-02-13]. Dostupné z: http://gs.statcounter.com/os-market-share/mobile/worldwide

[2] Warren, T.: Microsoft finally admits Windows Phone is dead [online]. [cit. 2018-02-13]. Dostupné z: https://www.theverge.com/2017/10/9/16446280/microsoft-finally-admits-windows-phone-is-dead

[3] Thirumala, V.: Interaction Design patterns : iOS vs Android [online]. [cit. 2018-02-13]. Dostupné z: https://medium.com/@vedantha/interaction-design-patterns-ios-vs-android-111055f8a9b7

[4] Platform Architecture [online]. [cit. 2018-02-18]. Dostupné z: https://developer.android.com/guide/platform/index.html

[5] T-Mobile Unveils the T-Mobile G1 — the First Phone Powered by Android [online]. [cit. 2018-02-18]. Dostupné z: https://newsroom.t-mobile.com/news-and-blogs/t-mobile-unveils-the-t-mobile-g1-the-first-phone-powered-by-android.htm

[6] Rubin, A.: Where's my Gphone? [cit. 2018-02-13]. Dostupné z: https://googleblog.blogspot.cz/2007/11/wheres-my-gphone.html

[7] Shafirov, M.: Kotlin on Android. Now official [online]. [cit. 2018-02-18]. Dostupné z: https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official

[8] Schmucker, K.: Is it illegal to run OSX in a virtual machine? I remember Apple saying their OS was free? [online]. [cit. 2018-03-01]. Dostupné z: https://www.quora.com/Is-it-illegal-to-run-OSX-in-a-virtual-machine-I-remember-Apple-saying-their-OS-was-free

[9] Hlava, T.: Fáze a úrovně provádění testů [online]. [cit. 2018-03-01]. Dostupné z: http://testovanisoftwaru.cz/category/druhy-typy-a-kategorie-testu/#unit

[10] Mlejnek, J.: Testování aplikací [online]. [cit. 2018-03-01]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-SI1//_media/lectures/09/09.prednaskacb.pdf

[11] The State of the Octoverse 2017 [online]. [cit. 2018-03-01]. Dostupné z: https://octoverse.github.com/

[12] Github Pricing [online]. [cit. 2018-03-01]. Dostupné z: https://github.com/pricing/team

[13] Bitbucket Pricing [online]. [cit. 2018-03-01]. Dostupné z: https://bitbucket.org/product/pricing

[14] Gitlab Pricing [online]. [cit. 2018-03-24]. Dostupné z: https://about.gitlab.com/pricing

[15] Gitlab Feature Comparison [online]. [cit. 2018-03-24]. Dostupné z: https://about.gitlab.com/pricing/self-hosted/feature-comparison/

[16] Do you support Bitbucket or GitLab? [online]. [cit. 2018-03-24]. Dostupné z: https://circleci.com/docs/1.0/faq/#do-you-support-bitbucket-or-gitlab

[17] Circle CI Linux pricing [online]. [cit. 2018-03-24]. Dostupné z: https://circleci.com/pricing/#build-linux

[18] Circle CI macOS pricing [online]. [cit. 2018-03-24]. Dostupné z: https://circleci.com/pricing/#build-os-x

[19] Kawaguchi, K.: Jenkins 1.396 released [online]. [cit. 2018-03-24]. Dostupné z: http://jenkins-ci.361315.n4.nabble.com/Jenkins-1-396-released-td3257106.html

[20] Dumay, J.: Say hello to Blue Ocean 1.0 [online]. [cit. 2018-04-02]. Dostupné z: https://jenkins.io/blog/2017/04/05/say-hello-blueocean-1-0/

[21] HockeyApp is now free for all developers [online]. [cit. 2018-04-02]. Dostupné z: https://hockeyapp.net/blog/2017/12/19/hockeyapp-is-now-free-for-all-developers.html

[22] AppBlade Pricing [online]. [cit. 2018-04-02]. Dostupné z: https://appblade.com/pricing

[23] Sonar Pricing [online]. [cit. 2018-04-02]. Dostupné z: https://www.sonarqube.org

[24] Sonar Cloud Pricing [online]. [cit. 2018-04-02]. Dostupné z: https://about.sonarcloud.io/sq/

[25] Driessen, V.: A successful Git branching model [online]. [cit. 2018-04-02]. Dostupné z: http://nvie.com/posts/a-successful-git-branching-model/

[26] Gitlab Requirements [online]. [cit. 2018-04-02]. Dostupné z: https://docs.gitlab.com/ce/install/requirements.html

[27] Gaudin, O.: SonarQube Requirements [online]. [cit. 2018-04-02]. Dostupné z: https://docs.sonarqube.org/display/SONAR/Requirements

[28] GitLab Installation [online]. [cit. 2018-04-02]. Dostupné z: https://about.gitlab.com/installation/#ubuntu

[29] Install GitLab Runner using the official GitLab repositories [online]. [cit. 2018-04-02]. Dostupné z: https://docs.gitlab.com/runner/install/linux-repository.html

[30] Android Studio downloads [online]. [cit. 2018-04-10]. Dostupné z: https://developer.android.com/studio/#downloads

[31] Gaudin, O.: Installing the Server [online]. [cit. 2018-04-10]. Dostupné z: https://docs.sonarqube.org/display/SONAR/Installing+the+Server

[32] API: Apps [online]. [cit. 2018-04-10]. Dostupné z: https://support.hockeyapp.net/kb/api/api-apps

[33] Registering Runners [online]. [cit. 2018-04-10]. Dostupné z: https://docs.gitlab.com/runner/register/index.html

[34] Build local unit tests [online]. [cit. 2018-04-10]. Dostupné z: https://developer.android.com/training/testing/unit-testing/local-unit-tests

[35] Test UI for a single app [online]. [cit. 2018-04-10]. Dostupné z: https://developer.android.com/training/testing/ui-testing/espresso-testing

Příloha **A**

Seznam použitých zkratek

EULA End-User License Agreement

IDE Integrated Development Environment

UI User Interface

SDK Software Development Kit

GB Gigabyte

RAM Random-access memory

URL Uniform Resource Locator

KVM Kernel Virtual Machine

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `apk` ..... adresář se spustitelnou Android aplikací
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `commands` ..... příkazy použité pro nastavení služeb
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF