FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	ElateMe - QA v multiplatformních aplikacích
Student:	Tomáš Grofek
Vedoucí:	Ing. Petr Pauš, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Cílem práce je analyzovat, navrhnout a realizovat komplexní zajištění QA služby, jež je provozována na webu a platformách Android a iOS komunikující s API. Proveďte konkrétně pro službu ElateMe.

1) Proveďte analýzu rizik změnových požadavků jednotlivých komponent a vliv na službu ElateMe.
2) Navrhněte:
 - procesy pro zabezpečení QA,
 - testovací prostředí, využijte Docker a Gitlab,
 - API testy a výkonnostní testy API.
3) Implementujte:
 - API testy,
 - automatizujte testy dle navržených procesů.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 4. ledna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

ElateMe - QA v multiplatformních aplikacích

Tomáš Grofek

Katedra Softwarového Inženýrství
Vedoucí práce: Ing. Petr Pauš, Ph.d

14. května 2018

Poděkování

Chtěl bych poděkovat vedoucímu práce panu Petru Paušouvi za odborné rady a vedení při psaní této práci. Mé velké díky také patří panu Michalu Maňenovi, vedoucímu projektu ElateMe, za vedení praktické části práce a velmi cenné konzultace, které mi byli významným zdrojem informací. Závěrem bych ještě poděkoval rodině, přátelům a všem ostatním, kteří mne podporovali při mém studiu.

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, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 14. května 2018 .....

Odkaz na tuto práci

Grofek, Tomáš. *ElateMe - QA v multiplatformních aplikacích.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce se zabývá problematikou zajištění kvality v multiplatformních aplikacích konkrétně pro službu ElateMe. V úvodní kapitole jsou analyzována rizika při integraci změnových požadavků a jejich dopady na jednotlivé komponenty služby ElateMe. Dále jsou v práci navrhnuty postupy pro prevenci analyzovaných rizik. Při návrhu postupů je kladen důraz na jejich efektivitu a nízké náklady na provoz. Z velké části se práce zabývá automatizací procesu vývoje software a automatizovaným testováním. V praktické části práce jsou implementovány a automatizovány testy funkcionality API. Navržené postupy mohou být předlohou pro obdobné procesy v ostatních multiplatformních aplikacích.

Klíčová slova quality assurance, multiplatformní aplikace, continuous integration, softwarové testování, automatizované testování, testovaní API, ElateMe, Gitlab, Docker

Abstract

This thesis deals with quality assurance in multiplatform applications specifically for ElateMe. The introductory chapter analyzes the risks of integrating change requirements and their impacts on individual ElateMe components. Furthermore, there are proposed procedures for the prevention of the analyzed risks. Designing procedures emphasizes their effectiveness and low implementation costs. For the most part, the thesis deals with automation of the development process and automated testing. In the practical part of the thesis are implemented and automated API functionality tests. Proposed procedures can be a model for similar processes in other multiplatform applications

Keywords quality assurance, multiplatform application, continuous integration, software testing, automated testing, API testing, ElateMe, Gitlab, Docker

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Dnešní svět si bez aplikací dokáže představit jen málokdo. Díky vysoké poptávce na trhu jsou stále vytvářeny nová zařízení, která dokáží aplikace spustit. Různorodost těchto zařízení přináší obtíže pro vývoj univerzálních aplikací, které by dokázaly fungovat na všech platformách. Proto jsou vyvíjeny multiplatformní aplikace.

Současný vývoj takovýchto aplikací vyžaduje náročnější koordinaci vývojářských týmů. Z tohoto důvodu se u multiplatformních aplikací ztěžuje i ověřování kvality produktu. K testování jednotlivých komponent přibývá i nutnost testování aplikace jako celku, a také se porucha nebo změna na jedné z platforem může projevit na jiné platformě.

Jelikož jednotlivé celky multiplatformní aplikace jsou si velmi podobné, vyvstává otázka, zda by testování těchto aplikací šlo zjednodušit, než pro každou platformu vyvíjet plnohodnotné testy. Analýza a návrh těchto postupů má potenciál ušetřit nemalé finanční obnosy na testování těchto aplikací.

ElateMe

ElateMe je crowdfundingová služba, která umožňuje vybírat finance na pořizování darů. Skutečnost, že je ElateMe dostupná na webu a mobilních telefonech s operačními systémy Android a iOS, činí tuto službu typickým zástupcem multiplatformních aplikací.

Cíl práce

Úkolem této práce je analyzovat rizika změn jednotlivých komponent, navrhnout postupy zajištění kvality u dalšího vývoje služby ElateMe a část těchto postupů realizovat prakticky. Tyto postupy však mají potenciál být použity pro většinu multiplatformních aplikací.

Motivace

Při řešení problematiky zajištění kvality služby ElateMe jsme s testovacím týmem nenašli žádné uspokojivé shrnutí novodobých možností testování těchto aplikací, proto jsem si vybral tuto práci.

Kapitola 1

Analýza

V této kapitole nejdříve vysvětlím, co jsou to multiplatfomní aplikace, poté rozeberu problematiku vývoje multiplatformích aplikací a na základě toho identifikuji největší rizika, která budou hrozit při integraci změn do projektu. Výstup této kapitoly je podkladem pro návrh postupů zajištění kvality ve vývoji služby ElateMe.

1.1 Platformy

Termínem výpočetní platforma[1, 2] se rozumí určitá množina hardwarových a softwarových technologií, která umožňuje spouštění aplikací. Každá taková platforma nabízí rozhraní pro vývoj aplikací (API – Application programing interface), které umožňuje platformu ovládat a používat její (často specifickou) funkcionalitu.

Z důvodu různorodosti platforem neexistuje žádná norma, která by unifikovala jejich rozhraní, proto aplikace vyvinuté pro jednu platformu nejsou většinou spustitelné na jiné platformě. Tento důvod byl hlavním podnětem pro vznik multiplatformních aplikací.

V oblasti IT je velmi bežnou praxí vyvíjet platformy, které využívají jiné platformy, za účelem přidání nové funkcionality. Velmi časté jsou platformy, které poskytují jednotné vývojové rozhraní pro více platforem nižší úrovně. Na platformy lze pohlížet s různou mírou abstrakce. Od základních kombinací hardwarových součástek až po robustní softwarové platformy, které shrnují funkcionalitu více platforem na nižší úrovni abstrakce. Novodobé webové prohlížeče již také splňují definici výpočetní platformy.

Pro vývoj aplikací jsou nejvíce využívanými platformami operační systémy (OS). OS poskytují jednotné aplikační rozhraní pro vícero kombinací hardwaru. Pro projekt ElateMe jsou to konkrétně platformy iOS, Android a webový prohlížeč.

1.2 Multiplatformní aplikace

Rivalita dodavatelů platforem a odlišnost jejich vývojových rozhraní způsobuje velké potíže vývojářům softwarových aplikací. Fakt, že aplikace vyvinuta pro jednu platformu nejde spustit na jiné platformě, velmi snižuje počet potencionálních uživatelů a hlavně výši potenciálního zisku. Efektivním řešením tohoto problému je vývoj multiplatformních aplikací[3, 4].

Multiplatformní aplikace je definována jako aplikace, která je spustitelná na více než jedné platformě. Tyto aplikace jsou vyvíjeny z důvodu, aby se nabízená funkcionalita dostala k co nejvíce cílovým uživatelům a zároveň je neomezovala ve výběru zařízení, která dokáží aplikaci spustit. Takové aplikace se často skládají z několika aplikací, každá určená pro danou platformu, které se jeví jako jedna tatáž aplikace. Každá tato aplikace poskytuje stejnou funkcionalitu. Tyto dílčí aplikace jsou klientskými komponentami výsledné multiplatformní aplikace.

1.2.1 Klientské komponenty

Koncoví uživatelé u aplikací nejvíce oceňují jejich uživatelskou přívětivost a vzhled. Přesně tohle zařizují klientské komponenty. Většina uživatelů přijde do styku pouze s těmito komponentami.

Hlavním úkolem klientských komponent je nabídnout funkcionalitu aplikace uživateli. Důraz je kladen hlavně na vysokou podobnost vzhledu všech aplikací. Tímto se snižuje dezorientovanost uživatele při používání aplikace na více platformách. Zároveň by každá komponenta měla poskytovat stejnou funkcionalitu, aby uživatelé nebyli nuceni preferovat určitou platformu.

1.2.2 Serverová komponenta

U většiny aplikací je vyžadováno pamatování stavu a uchovávání dat. Takové aplikace se nazývají tzv. stavové. Pokud by se tato data uchovávala zvláště v každé klientské komponentě, musela by být sdílena napříč komponentami, aby nedocházelo k jejich desynchronizaci. Aby se vyhnulo implementování složité komunikace mezi těmito komponentami za účelem synchronizace dat, je součástí těchto aplikací serverová komponenta.

Přestože tato komponenta neprezentuje funkcionalitu aplikace výsledným uživatelům, je nejdůležitější součástí každé stavové aplikace. Svou funkcionalitou jde serverová komponenta bez nadsázky přirovnat k srdci aplikace. Jedním z jejich úkolů je uchovávat data a poskytovat je klientským komponentám pomocí API tak, aby byly synchronizované a aktuální. Mezi další úkoly se řadí autorizace, autentikace uživatelů a řízení, synchronizace komponent.

1.2.3 Typy Multiplatformních aplikací z pohledu vývoje

V současné době lze klientské komponenty multiplatformních aplikací vyvíjet mnoha způsoby. Způsob, jakým jsou vyvinuty, ovlivňuje náklady na vývoj a technické vlastnosti aplikace, ale také může ovlivnit složitost jejího testování. V současné době existují tři základní typy vývoje multiplatformních aplikací: nativní aplikace, nativní multiplatformní aplikace a hybridní aplikace. V této kapitole rozvinu článek z blogu webové stránky krosapp [5].

Nativní Aplikace

Vývoj nativních aplikací je základním typem vývoje multiplatformních aplikací. Pro každou platformu je vyvinuta samostatná klientská komponenta za využití rozhraní dané platformy. Mezi obrovské výhody tohoto typu vývoje patří rychlost, spolehlivost a maximální možnosti přizpůsobení vyvíjené aplikace. ElateMe je vyvíjeno tímto typem vývoje multiplatformních aplikací.

Nativní multiplatformní aplikace

> „Nativní multiplatformní aplikace je napsána pomocí nástroje, který umožňuje její překlad do nativních jazyků a tím umožňuje její běh na vícero platformách s jednotným sdíleným zdrojovým kódem. Na rozdíl od následujících hybridních aplikací je výsledná zkompilovaná aplikace 100% nativní – využívá prostředků přímo dané konkrétní platformy a chová se jako klasická nativní aplikace.“[5]

Tento způsob vývoje snižuje požadavky na rozmanitost technické odbornosti vývojového týmu a vyvinuté aplikace se velmi podobají nativním. Na druhou stranu prináší omezení pro využití rozhraní daných platforem a uživatelské rozhraní (User interface – UI) se často musí dělat pro každou platformu zvlášťě. Za největší nedostatek tohoto typu vývoje považuji omezený počet nástrojů, které ho umožňují, ale také fakt, že tyto nástroje podporují jen dané množství platforem, na které lze aplikaci vyvinout.

Hybridní aplikace

> „Hybridní aplikace vycházejí z principu interpretace HTML5 kódu ve speciálních „kontejnerech“, které se tváří jako nativní aplikace. Ve skutečnosti se tak vlastně jedná o webovou stránku, která se ale zobrazuje uvnitř samostatné aplikace. Nepoužívá nativní ovládací prvky dané platformy (tlačítka, výběrové prvky apod.) ale vlastní HTML prvky, které se liší od těch nativních na jednotlivých platformách.“[5]

Snad jedinou výhodou tohoto postupu je požadavek na nízkou technickou odbornost vývojového týmu. K vyvinutí aplikace tímto způsobem stačí znalost HTML, CSS a Javascriptu. Za 100% sdílenost zdrojového kódu se platí složitějším přizpůsobováním UI pro jednotlivé platformy. K nevýhodám se řadí mnohem nižší výkon v porovnání s nativními aplikacemi a velmi omezený přístup k rozhraním platforem. Jelikož je aplikace v podstatě běžící webová stránka uvnitř nativní aplikace, může vypadat a chovat se jinak, než by se očekávalo od nativní aplikace.

1.3 Rizika změnových požadavků na komponenty

Většina aplikací je vyvíjena, aby přinesla usnadnění určitých činností uživatelům. Pokud jsou uživatelé s funkcionalitou aplikace spokojeni, rádi ji doporučí známým, kladně ji hodnotí v různých recenzích a podobnými způsoby dávají najevo svou spokojenost s danou aplikací. Tímto se uživatelská komunita aplikace rozrůstá, což je základním požadavkem na úspěch aplikace. Počet uživatelů aplikace je základním stavebním kamenem pro její komerční využití. Je to jednoduchá přímá úměra: čím více uživatelů, tím větší zisky aplikace produkuje.

Pokud uživatelé s funkcionalitou aplikace nejsou spokojeni, určitě ji nedoporučí, ba dokonce mohou ostatní potenciální uživatele odrazovat od použití aplikace. V tomto případě klesá jak počet uživatelů, tak i zisky. Díky těmto skutečnostem je spokojenost uživatelů s aplikací velmi důležitá pro úspěch aplikace a je nutno ji držet na vysoké úrovni.

V současné době je žádané, aby se aplikace s postupem času neustále vylepšovala, tudíž je nutno integrovat změny do již stávajících funkcionalit aplikace. Tyto změny s sebou nesou rizika a jejich dopady, které by mohly poškodit stávající funkcionalitu aplikace a tím snížit spokojenost uživatelů. Pokud by tato rizika nastala, lidé by mohli ztratit důvěru v aplikaci, která se velmi těžce získává zpět. Ztráta spokojenosti uživatelů by mohla službě ElateMe přivodit kritické existenční problémy. Mimo ztrátu uživatelů, by nastalá rizika znamenala i náklady na jejich opravu, které jsou v daném průmyslovém odvětví vysoké. Je nutné tato rizika identifikovat, analyzovat a poté na základě analýzy podniknout kroky k jejich zabezpečení, nebo alespoň minimalizaci.

Jak již bylo zmíněno, komunikace každé klientské komponenty probíhá pouze mezi ní samotnou a API serverové komponety. Tento fakt zamezuje propagaci případných chyb vzniklých integrací změnových požadavků v klientské komponentě do ostatních komponent. Navíc jsou klientské komponenty zodpovědné pouze za prezentaci funkcionality výsledným uživatelům, tudíž výskyt případné chyby neporuší důležité funkcionality aplikace v jiných komponentách. Na druhou stranu uživatelé přichází do styku výhradně s klientskými komponentami, takže i chyby vzniklé v klientských komponentách by mohly znamenat výrazný úbytek spokojenosti.

Serverová komponenta si vyměňuje data se všemi komponentami. Případná chyba by se mohla propagovat dále do klientských komponent, kde by mohlo dojít rovněž k narušení funkcionality. Pokud by tato událost nastala a funkcionalita celé služby by byla narušena, znamenalo by to značný pokles spokojenosti zákazníků. Navíc je tato komponenta zodpovědná i za uchovávání dat, spravování plateb a financí, autorizaci a autentikaci uživatelů, aj. Případná chyba by mohla způsobit mnoho nepříjemných situací, které by jistě vedly k závažnému poklesu spokojenosti uživatelů. To by mohlo mít kritické následky pro službu ElateMe.

Rizika ovšem nepředstavuje pouze možnost vzniku chyby, ale také nástroje, které jsou používány k vývoji a chodu aplikace. Některá rizika dokonce nemůže ovlivnit vývojový tým ElateMe, ale jsou závislé na funkcionalitách třetích stran.

1.3.1 Identifikace a analýza rizik

Jak již bylo zmíněno, nejrizikovější jsou požadavky na změnu funkcionality serverové komponenty. Pro úspěšný návrh minimalizace těchto rizik je nutné rizika prve identifikovat a analyzovat. U každého identifikovaného rizika je nutné určit jeho možné příčiny a dopady na službu ElateMe. U příčin se dále analyzuje pravděpodobnost jejich výskytu, u dopadů zase jejich závažnost pro chod aplikace.

1.3.1.1 Prodlení opravy chyb

Každý dělá chyby. Ani u softwarových vývojářů tomu není jinak. Pokud se podaří chybu detekovat ještě před vypuštěním do produkčního prostředí, nezpůsobí téměř žádné škody. Přinejhorším bude nutno odložit nasazení aplikace na produkční prostředí a odstranění chyby bude stát náklady navíc.

Pokud se ale chybu nepodaří detekovat zavčas a dojde k jejímu nasazení do produkčního prostředí, chyba přijde do kontaktu s uživateli. Některé takové chyby zapříčiní částečnou nefunkčnost některých funkcionalit aplikace, jiné ovšem mohou poškodit veškerou funkcionalitu a tím udělat aplikaci nepoužitelnou. V každém případě je nutno chybu co nejrychleji odstranit, protože po dobu nefunkčnosti aplikace stoupá počet uživatelů, kteří přišli do styku s danou chybou.

Po nahlášení chyby je nutné ji neprodleně detekovat, opravit a vydat opravenou verzi aplikace. Detekce a oprava chyby je u všech komponent podobně časově náročná. Tento čas je nezbytný pro opravu chyby. Co se však u jednotlivých komponent v časové náročnosti liší, je vydání nové verze s opravenou funkcionalitou.

U serverové a webové komponenty je čas potřebný pro vydání nové verze minimální, protože obě tyto komponenty jsou spuštěné na serverech, které

vlastní provozovatel služby. Pro vydání nové verze není potřeba žadných kroků navíc, než je nasazení dané verze na servery.

U klientských komponent na platformách Android a iOS tomu je jinak, protože pro distribuci k uživatelům je využito nástrojů třetích stran. U Android aplikace je k distribuci využito Google Play[6], u iOS aplikace je to App Store[7]. Tito poskytovatelé aplikací pro dané platformy jsou uživateli pravděpodobně nejvíce navštěvovanými a používanými pro dané platformy, tudíž je u nich největší poptávka po aplikacích. Pro zajištění kvality jimi nabízených aplikací, musí aplikace splňovat určité požadavky. Před umístěním do nabídky, musí každá akce projít manuálním schvalovacím procesem. Tento fakt značně prodlužuje čas, který je potřeba k vydání nové verze.

Proces kontroly aplikací u Google Play není příliš striktní. V tomto procesu je převážně kontrolováno, zda aplikace neporušuje smluvní podmínky pro distribuci aplikace touto službou. Tento proces trvá přibližně několik hodin.

U App Store je tento proces komplikovanější. Ke kontrole dodržení smluvních podmínek se navíc provádí i letmé testování funkcionality. Tímto postupem se zajistí, že pomocí App Store budou distribuovány pouze kvalitní a funkční aplikace. Díky tomuto faktu se minimalizuje riziko distribuce aplikace s chybou, bohužel však na úkor času potřebného pro schválení aplikace. Tento proces může trvat až několik dní.

App Store nabízí možnost urychlení procesu schválení aplikace při splnění daných podmínek, které poté proběhne v řádu hodin. Počet těchto možností je však omezen a není garantováno, že urychlení procesu bude vyhověno. O urychlení procesu lze zažádat pouze pokud nová verze obsahuje opravu závažné chyby, nebo je spojená s událostí, která je časově omezená. Proti zneužití této funkcionality je nutno uvést kroky k reprodukci chyby ve stávající aplikaci, nebo přiložit popis události, datum jejího konání a její spojitost s distribuovanou aplikací.

Příčinou tohoto rizika je proniknutí chyby do produkčních prostředí komponent. Dopadem je nedostupnost funkcionality aplikace na dobu, jejíž délka je pro každou platformu odlišná.

1.3.1.2 Nekompatibilita klientských aplikací s daty poskytovanými API

Funkcionalita klientských komponent je závislá na datech, které získává od API serverové komponenty. Změna datových typů, struktury nebo obsahu v těchto datech by mohla způsobit poruchu funkcionality klientských komponent. Tato změna se týka pouze serverové komponenty, tudíž jeho příčinou je požadavek na změnu dat poskytovaných API.

Z důvodu různorodosti platforem může být dopad na jednotlivé klientské komponenty odlišný. Váha dopadů je úzce spjata s vlastnostmi použitých programovacích jazyků a s konkrétní implementací komponent.

Webová komponenta je napsána v programovacím jazyce Javascript. Díky dynamickému typování případné změny v datech pravděpodobně nezpůsobí úplné vyřazení funkcionality komponenty. Jednoduché změny datových typů (např. přetypování integeru na double), nebo některé změny ve struktuře (např. přidání nového parametru do objektu v JSON) nepoškodí funkcionalitu vůbec. Dopady dalších změn se liší dle konkrétní implementace v klientské komponentě. Některé mohou „pouze“ zobrazit špatná data, jiné zase mohou vést ke špatnému vykreslování jednotlivých prvků uživatelského rozhraní, nebo k porušení jejich funkcionality. Důsledek u webové komponenty je tedy částečné znepřístupnění funkcionality.

U klientských komponent na platformách Android a iOS jsou dopady mnohem vážnější. Android aplikace je napsána v programovacím jazyce Java, iOS ve Swiftu. Oba tyto jazyky jsou silně typované, takže i jednoduché změny datových typů zapříčiní selhání aplikace, jelikož aplikace nebude schopna namapovat tyto datové typy na vlastní objekty. Stejný výsledek budou mít i sofistikovanější změny dat.

1.3.1.3 Snížení výkonnosti API

Rychlá odezva je v současné době vyžadována od všech aplikací. Zpomalení odezvy bývá pro naprostou většinu uživatelů přinejmenším nepříjemné a vede ke snížení jejich spokojenosti, proto je tento jev naprosto nežádoucí. Ke zpomalení odezvy aplikace může docházet z mnoha příčin. Část z nich bohužel vývojáři aplikací nemohou ovlivnit (např. pomalé internetové připojení, nebo nedostatečný výpočetní výkon zařízení, na kterém uživatel aplikaci spouští). Ty ostatní jsou zapříčiněny chybami v aplikaci.

Jelikož v klientských komponentách není vyžadováno žádných náročných výpočtů, je pravděpodobnost zpomalení funkcionality ze strany klientských komponent téměř nulová. Jinak je tomu u serverové komponenty, kde se pro získání dat používají nejrůznější algoritmy a dotazy nad databází. Z příčiny implementace příliš pomalého algoritmu, nebo použití nevhodného, neefektivního dotazu pro získání dat z databáze může dojít k rapidnímu zvýšení doby odezvy API. V krajním případě může dojít i k úplnému zahlcení služby a jejímu následnému selhání.

Ke zpomalení služby mohou vést i jiné příčiny. Nadměrné zapisování logu aplikace se také velmi podepisuje na rychlosti provedených postupů, jelikož zápis na pevný disk je mnohonásobně pomalejší v porovnání s výpočetními operacemi. Také s přibývajícími záznamy v databázi se úměrně prodlužuje doba vyhodnocení dotazů vykonaných pro získání dat.

Snížení výkonnosti API může být způsobeno mnoha příčinami. Ve všech případech je však důsledkem většinou „pouze“ zpomalení odezvy, které nemá vliv na funkcionalitu aplikace. I přesto však vede ke snížení spokojenosti uživatelů a vyplatí se vykonat opatření k minimalizaci tohoto rizika. V krajních

situacích může být serverová komponenta zahlcena natolik, že dojde k jejímu úplnému selhání, a následné nedostupnosti aplikace.

1.3.1.4 Únik citlivých dat

Služba ElateMe zpracovává mimojiné i citlivé uživatelské údaje. Pro práci s těmito údaji je vyžadováno adekvátní zabezpečení. Zásahy do zabezpečení by mohli vést k úniku těchto dat, což může vyústit v mnoho nepříjemností.

Klientské komponenty pouze prezentují uživateli data, která získají od API serverové komponenty, proto je únik citlivých dat skrze klientské komponenty prakticky nemožný. Za správu citlivých údajů je zodpovědná serverová komponenta. Veškerá citlivá data musí být poskytována pouze autorizovaným osobám. Změnami v zabezpečení serverové komponenty, by mohly vzniknout chyby, které by umožňovaly přístup k datům neautorizovaným uživatelům. Hackeři a zručnější jedinci je mohou poměrně jednoduše získat i přesto, že tato data nebudou běžným uživatelům přístupná.

Pokud by tato příčina nastala a uživatelé by se o ní dověděli, nejenže by to způsobilo značnou nedůvěru v seriózност služby, ale důsledkem by byly i soudní žaloby. Pokles uživatelské spokojenosti by byl v tomto případě kritický.

1.3.1.5 Únik finančních prostředků

Hlavní funkcionalitou služby ElateMe je zprostředkování výběru finančních prostředků na dárky a poté jejich předání vlastníkovi přání. Správa financí probíhá plně automatizovaně v režii serverové komponenty. Tato správa musí být rovněž řádně zabezpečena, aby nemohlo dojít k jejímu narušení.

Jelikož klientské komponenty nenesou žádnou zodpovědnost za správu vybraných financí, tohle riziko se týká pouze serverové komponenty. Změnovým požadavkem na serverovou komponentu, obzvláště požadavkem na změnu zabezpečení nebo správu financí, by mohla vzniknout bezpečnostní díra, skrze kterou by mohli hackeři ovlivňovat správu financí. Důsledkem by sice nebylo porušení funkcionality nebo snížení uživatelské spokojenosti, ale vysoké finanční ztráty.

1.3.1.6 Narušení funkcionality uživatelskými vstupy

Všechny novodobé aplikace interagují s jejich uživateli. Jedním ze způsobů této interakce je sběr dat zadaných uživateli, jejich zpracování, uložení a další využití. Nedostatečné ošetření dat přijímaných od uživatelů by mohlo mít mnoho vážných důsledků.

Protože částečné ošetření uživatelských vstupů probíhá i na klientských komponentách, jsou příčinou tohoto rizika i změny provedené v klientských aplikacích. Tyto uživatelské vstupy jsou dále předávány API serverové komponenty, kde je nutné jejich další ověření. Hlavní zodpovědnost za ošetření uživatelských vstupů tudíž nese serverová komponenta.

Ukládání záznamů obrovské velikosti by mohlo zahltit databázi, či dokonce vyčerpat veškeré dostupné místo na datovém úložišti. Stejný dopad by mohlo mít ukládání binárních souborů nadměrné velikosti. Tyto příčiny by vedly ke zpomalení, či dokonce k selhání a následné nedostupnosti serverové komponenty. Dalším dopadem je narušení funkcionality klientských komponent při zpětné prezentaci uložených dat. Všechny tyto dopady opět ústí ve znatelné snížení uživatelské spokojenosti. Váha dopadů se opět liší v závislosti na platformě klientské komponenty.

Na webové aplikaci načítání dat nadměrné velikosti způsobí pouze zpomalení načtení těchto dat. V krajním případě může dojít k vyčerpání přidělené operační paměti a následnému selhání aplikace. V klientských komponentách pro platformy Android a iOS jsou dopady podobné, s tím rozdílem, že je mnohem pravděpodobnější vyčerpání přidělené operační paměti a selhání aplikace.

Neošetření uživatelských vstupů může rovněž zapříčinit uložení škodlivých dat. Tato data poté mohou narušit funkcionalitu webové komponenty, jelikož mohou být prezentována a spuštěna jako HTML nebo javascriptový kód. Tato příčína umožňuje napadení služby metodou cross-site scripting (XSS).

1.3.1.7 Narušení uživatelských pravomocí

Ve službě ElateMe může uživatel vystupovat v mnoha rolích (nepřihlášený uživatel, přihlášený uživatel, dárce, atd.). Každá tato role má své specifické pravomoce, které jim umožňují vykonávat určité akce a zajišťují přístup k určitým datům a funkcionalitám.

Změna v uživatelských rolích a jejich oprávnění by mohla zapříčinit narušení funkcionality spjaté s danými rolemi. Důsledkem čehož by uživatelé mohli neoprávněně využívat některé funkcionality, nebo získat neautorizovaný přístup k datům. Rovněž by mohlo dojít k znepřístupnění určitých funkcionalit daným uživatelským rolím.

Jelikož se toto riziko týká uživatelských rolí, je málo pravděpodobné, že dopad postihne všechny uživatele. Dopad tedy pravděpodobně naruší funkcionalitu jen některým uživatelským rolím bez ohledu na platformu klientské komponenty. I přesto tento důsledek znamená pokles uživatelské spokojenosti.

1.3.1.8 Porušení současné funkcionality

Změny ve funkcionalitě aplikace jsou velmi časté. Jelikož tyto funkcionality spolu často souvisí, může se změna projevit i na jiných funkcionalitách. Implementace nové, odebírání staré, či změna stávající funkcionality může zapříčinit narušení jiné, již implementované a plně funkční funkcionality.

Váha dopadu tohoto rizika se liší pro každou platformu. Porušení stávající funkcionality u klientských komponent mívá zpravidla mírnější dopad, než chyba u serverové komponenty. Zatímco u klientských komponent se chyba projeví pouze na funkcionalitě dané komponenty, chyba v server API může

narušit funkcionalitu všech komponent. Nedostupnost funkcionality opět znamená pokles uživatelské spokojenosti. Navíc je nutné chybu odstranit, což znamená další náklady.

1.3.1.9 Nekompatibilita verzí

V technice se postupem času vše vyvíjí. Jinak tomu není i u platforem, proto jsou v dnešní době téměř všechny platformy dostupné v několika verzích, jejichž parametry, aplikační rozhraní a chování se může (a často tomu tak je) lišit.

Jen málokterý uživatel používá vždy nejnovější verze platforem, proto je nutné jich podporovat co nejvíce pro docílení maximální dostupnosti aplikace. Podpora všech verzí platforem je ekonomicky nevýhodná, takže je vhodné podporovat jen ty, které jsou pro službu výhodné. To se dá určit pomocí analýzy využívání platforem verzí, která určí procentuální podíl uživatelů využívající danou verzi platformy ze všech uživatelů, využívající danou platformu. Podporují se poté jen ty verze, které mají vyšší procento uživatelů než je stanovená mez provozovateli služby. Pro ElateMe je tato mez stanovená na 5 %.

Pro podporu vybraných verzí je nutno distribuovat klientské komponenty ve verzích kompatibilních s těmito verzemi. Navíc tyto verze nemusí být kompatibilní s aktuální verzí API serverové komponenty, takže musí být distribuováno i více verzí API, které budou kompatibilní se všemi distribuovanými verzemi klientských komponent. To s sebou přináší riziko, že nové změny v komponentách naruší tuto kompatibilitu.

U každé nové verze klientské komponenty je nutné zajistit, aby byla kompatibilní alespoň s jednou (často tou nejnovější) podporovanou verzí API. Pokud by tomu tak nebylo, s vysokou pravděpodobností by byla narušena část funkcionality této komponenty.

Naopak u každé změny v API serverové komponenty je vyžadována její kompatibilita se všemi verzemi klientských komponent, které byli kompatibilní s předchozí verzí API. Pokud tento požadavek nebude splněn, funkcionalita některých verzí klientských komponent bude narušena.

1.3.2 Shrnutí rizik

Z analýzy je patrné že nejvíce příčin rizik pochází od změnových požadavků serverové komponenty a zároveň jsou jejich dopady na službu ElateMe kritičtější. Na druhou stranu se příčiny velmi často projevují právě na klientských komponentách, kde se dostávají do styku s uživateli.

Obecně lze říci, že přičiny všech těchto rizik jsou chyby v aplikaci vzniklé integrováním změn. Podobně naprostá většina dopadů ústí v pokles uživatelské spokojenosti, což má negativní vliv na úspěch služby. Pro prevenci těchto rizik je tedy vhodné využít procesů, které minimalizují riziko výskytu chyb v nově integrovaných změnách.

V následujících tabulkách 1.2 1.3 1.4 1.5 si čtenář může prohlédnout shrnutí rizik. Každá tabulka obsahuje rizika týkající se (příčinou nebo dopadem) jedné komponenty. U všech těchto rizik najdete vyhodnocení pravděpodobnosti výskytu (sloupec označený „P“) a váhu jejich dopadu (sloupec označený „V“). Hodnoty pravděpodobnosti výskytu a dopadu nalezne čtenář v tabulce 1.1.

Tabulka 1.1: Tabulka hodnot pravděpodobnosti a dopadu

Hodnota	Pravděpodobnost (P)	Váha dopadu (V)
1	téměř nemožné	zanedbatelný
2	nepravděpodobné	nevýznamný
3	možné	střední
4	pravděpodobné	významný
5	téměř jisté	kritický

Tabulka 1.2: Tabulka rizik spjatých s komponentou pro platformu iOS

#	Příčina	P	V	Dopad
#1	Nahrání verze s chybou do App Store	2	4	Nedostupnost aplikace na několik dní
#2	Nekompatibilita se změnou dat z API	3	4	Selhání aplikace, 100 % nepoužitelnost
#3	Přijmutí vadných dat z API	2	4	Vyčerpání přidělené paměti, pád aplikace
#4	Nedostatečné ošetření uživatelských vstupů	2	1	Předání škodlivých dat API
#5	Porušení již implementované funcionality	3	3	Nepoužitelnost dané funkcionality
#6	Změny nekompatibilní s podporovanou verzí API	3	3	Nefunkčnost některých funkcionalit

Tabulka 1.3: Tabulka rizik spjatých s komponentou pro platformu Android

#	Příčina	P	V	Dopad
#1	Nahrání verze s chybou do Google Play	3	3	Nedostupnost aplikace na několik hodin
#2	Nekompatibilita se změnou dat z API	3	4	Selhání aplikace, 100 % nepoužitelnost
#3	Přijmutí vadných dat z API	2	4	Vyčerpání přidělené paměti, pád aplikace
#4	Nedostatečné ošetření uživatelských vstupů	2	1	Předání škodlivých dat API
#5	Porušení již implementované funcionality	3	3	Nepoužitelnost dané funkcionality
#6	Změny nekompatibilní s podporovanou verzí API	3	3	Nefunkčnost některých funkcionalit

Tabulka 1.4: Tabulka rizik spjatých s webovou komponentou

#	Příčina	P	V	Dopad
#1	Nahrání verze s chybou na produkční server	3	2	Nedostupnost aplikace po dobu uploadu funkční verze
#2	Nekompatibilita se změnou dat z API	2	3	Narušení funkcionality, rozladění vzhledu
#3	Přijmutí velkých dat z API	2	1	Delší doba načítání stránky
#4	Přijmutí vadných dat z API	2	5	Narušení vzhledu a funkcionality, umožnění XSS útoku
#5	Nedostatečné ošetření uživatelských vstupů	2	1	Předání škodlivých dat API
#6	Porušení již implementované funcionality	3	3	Nepoužitelnost dané funkcionality
#7	Změny nekompatibilní s podporovanou verzí API	3	3	Nefunkčnost některých funkcionalit

Tabulka 1.5: Tabulka rizik spjatých se serverovou komponentou

#	Příčina	P	V	Dopad
#1	Nahrání verze s chybou na produkční server	3	3	Nedostupnost aplikace po dobu uploadu funkční verze
#2	Změna dat poskytovaných API se kterou nejsou kompatibilní prezentační komponenty	3	5	Narušení nebo úplné znepřístupnění funkcionality všech prezentačních komponent
#3	Pomalý algoritmus, neefektivní dotaz nad databází, nadměrné logování	3	2	Prodloužení odezvy API
#4	Vyčerpání paměti, zahlcení služby vlivem příčin z #3	2	5	Selhání komponenty, nedostupná funkcionalita ostatních komponent
#5	Bezpečnostní díra	2	5	Únik citlivých dat
#6	Bezpečnostní díra	2	4	Únik finančních prostředků
#7	Nedostatečné ošetření uživatelských vstupů	2	3	Plýtvání s pamětí, propagace vadných dat do klientských komponent
#8	Chyba ve změně uživatelských pravomocí	2	4	Zpřístupnění neoprávněných funkcionalit, Znepřístupnění oprávněných funkcionalit
#9	Změny nekompatibilní s podporovaými verzemi klientských komponent	3	4	Nefunkčnost některých funkcionalit nekompatibilních komponent

Kapitola 2

Procesy pro zabezpeční QA

Analýza ukázala, že změnové požadavky s sebou přináší mnoho rizik, která by mohlá vyústit v kritický pokles uživatelské spokojenosti. Nelze se spoléhat na pravděpodobnost, že se rizika neprojeví. K jejich prevenci je nutno co nejvíce minimalizovat výskyt chyb.

V této kapitole jsou navrženy procesy pro zabezpečení QA (Quality assurance). Při návrhu procesů je kladen důraz především na prevenci všech identifikovaných rizik a na efektivnost těchto procesů z pohledu časové náročnosti.

2.1 Automatizovaný proces vývoje

V současné době jsou možnosti nástrojů pro vývojáře velmi rozsáhlé. Jelikož strojové vykonávání postupů šetří lidské prostředky, je jejich automatizace stále více využívána. Nástroje automatizace se stále vyvíjejí a množina automatizovatelných procesů, která je obsáhlá již v současné době, se stále rozrůstá.

Téměř všechny životní cykly procesů vývoje aplikací obsahují pět základních kroků. Jsou jimi analýza, návrh, implementace, testování, nasazení. Vizuální zobrazení posloupnosti těchto kroků čtenář nalezne na obrázku 2.1. Procesy se liší pouze v počtu iterací těchto kroků a délkou každé iterace. V současné době jsou velmi často používány agilní postupy[8], kde se tento cyklus v průběhu vývoje opakuje zpravidla vícekrát. Jeho automatizace může ušetřit značné časové i finanční prostředky.

Pro ElateMe (a multiplatformní aplikace obecně) je automatizace efektivní hlavně kvůli potřebě testování funkcionality a kompatibility více verzí aplikací, které je potřeba dělat opakovaně s každou nově začleněnou změnou. Pro automatizaci začlenění změn a částečné automatizované testování aplikace je využit proces Continuous Integraton (CI). Popis navrženého automatizovaného vývoje čtenář nalezne ve třetí kapitole.

Obrázek 2.1: Posloupnost základních kroků vývoje software

2.2 Automatizované testování

Před nasazením aplikace na produkční prostředí je nutno otestovat správnost její funkcionality. Automatizovaný proces vývoje software je skvěle doplňován automatizovanými testy. Ve většině případů jsou tyto testy nezbytné pro zajištění kvality vyvíjeného software.

Nesmírnou výhodou automatizovaných testů je jejich znovuspustitelnost, která nestojí prakticky žádné prostředky navíc. Na druhou stranu, aby testy splnily svůj účel, musí být precizně navržené a implementované tak, aby testovaly veškerou funkcionalitu. Návrh a implementace takových testů stojí mnoho času a úsilí, proto se automatizace nejvíce vyplatí u testů, které jsou prováděny opakovaně.

2.2.1 Unit testy

Unit testy (testování jednotek) slouží k testování základních funkcionalit zdrojového kódu. Cílem je otestovat funkcionalitu jednotlivých implementovaných procedur nezávisle na ostatních. Z tohoto důvodu je nutné, aby ke každé třídě a jejím procedurám ve zdrojovém kódu byly vykonány příslušné Unit testy. Tyto testy jsou zpravidla automatizované.

Implementací Unit testů funkcionalit je pověřen přímo programátor, který dané funkcionality implementoval. Tyto testy ověří a zajistí základní funkčnost zdrojového kódu, což navíc snižuje pravděpodobnost narušení komplexnějších funkcionalit služby. Případné chyby ve zdrojovém kódu jsou objeveny brzy, ještě než jsou využity jinými častmi zdrojového kódu. Ve spojení s ostatními testy slouží pro minimalizaci rizik spojených s objevením chyb v aplikaci a narušením jejích funkcionalit.

2.2.2 Akceptační testy

Narozdíl od Unit testů, které ověřují základní funkcionality zdrojového kódu, akceptační testy ověřují komplexnější funkcionality celé aplikace. U akceptačních testů je nutno otestovat veškeré definované funkční požadavky aplikace.

Toto testování bude vykonáváno formou regresních testů, které se spustí po každé integraci změny, aby bylo zaručeno nenarušení již implementovaných funkcionalit a zároveň se plně využilo benefitů automatizovaného testování. Funkční testování opět minimalizuje rizika spojená s objevením chyb a narušením funkcionality.

2.2.3 Výkonnostní testy

K prevenci rizik spojených se snížením výkonu a následném prodloužení odezvy API serverové komponenty je nutno implementovat výkonnostní testy. Od těchto testů je očekáváno měření výkonu API a jeho porovnávání s výkonem předchozích verzí. Popis navržených výkonnostních testů čtenář nalezne ve čtvrté kapitole.

2.3 Manuální testování

Jelikož ElateMe nabízí funkcionalitu, pro kterou by byla implementace automatizovaných testů velmi nákladná a nevýhodná, je třeba provádět i manuální testování. Touto funkcionalitou je správa bankovních převodů, která je automatizovaně netestovatelná zejména z důvodu prodlevy při vyřízení transakce.

Velkou výhodou manuálního testování je fakt, že narozdíl od automatizovaných testů nevyžaduje časově náročnou implementaci. Z tohoto důvodu je pro službu ElateMe momentálně nejspolehlivější formou testování, protože velká část automatizovaných testů ještě nebyla navržena a implementována. Manuální testování bude tedy mimo testy plateb suplovat zatím neimplementované automatizované testy.

2.4 Revize zdrojového kódu

Každá aplikace obsahuje často velmi obsáhlý zdrojový kód. Při vývoji software je nutno stávající zdrojový kód často měnit nebo na něj navázat, k čemuž je nutné se ve zdrojovém kódu zorientovat a pochopit jej. Při spolupráci na psaní zdrojového kódu více vývojáři se prakticky vždy značně snižuje jeho srozumitelnost a přehlednost.

Pro udržení srozumitelnosti a přehlednosti zdrojového kódu je proto nutno vyžadovat po programátorech dodržování stanovených jednotných pravidel pro jeho psaní. Mezi tato pravidla patří například jednotné pojmenovávání proměnných, odsazování kódu, vhodné komentování kódu, atd. Kontrola dodržení těchto pravidel probíhá pomocí revizí zdrojového kódu.

Srozumitelný a přehledný zdrojový kód přináší mnoho výhod. Orientace v takovém kódu je vždy mnohem rychlejší, a protože je tato činnost při vývoji vykonávána často, výrazně šetří čas vývojářů samotných. Další značnou výhodou je fakt, že nepochopení zdrojového kódu mnohdy vede k jeho nesprávnému použití a tedy i ke vzniku chyb. Udržování vysoké srozumitelnosti zdrojového kódu tedy mimojiné snižuje výskyt potencionálních chyb.

Pro revize zdrojového kódu již existuje mnoho nástrojů, které umožňují jejich vykonávání automatizovaně. V tomto případě je provedení revizí podstatně levnější a rychlejší.

Na druhou stranu revize zdrojového kódu prováděné osobou mají také své výhody. K jejich vykonávání bývají často pověřeni seniorní vývojáři s dlouhodobými zkušenostmi s vývojem v daném programovacím jazyce. Ti poté programátorům poskytují potřebnou odezvu z výsledku revize. Součástí této odezvy jsou často rady a nápady, které vedou ke zdokonalování vývojářů. Navíc manuální revize může lépe odhalit chyby v implementaci a nepochopení zdrojového kódu, které by strojově vykonaná revize nemusela odhalit.

2.5 Uživatelské testování

Jelikož úspěch aplikace silně závisí na spokojenosti jejich uživatelů, je vhodné aplikaci podrobit testování uživateli. Hlavním výstupem těchto testů je zhodnocení uživatelské přívětivosti (User experience – UX) uživatelského rozhraní aplikace přímo uživateli. Tyto výstupy pomáhají odhadnout spokojenost uživatelů s aplikací. Uživatelé, kteří ještě neznají plný rozsah funkcionalit a použitelnosti aplikace, ji často používají jinými způsoby než vývojáři a testeři. Proto je zde navíc pravděpodobnost, že budou odhaleny další chyby.

Při uživatelském testování několik uživatelů provádí na aplikaci úkoly, které jim zadá moderátor testů. Při těchto testech je nutné klást důraz na samostatnost uživatele, aby jeho chování nebylo nijak ovlivněno. Na základě těchto testů lze vyhodnotit intuitivnost aplikace.

Obecně u multiplatformních aplikací je tento typ testování poměrně výhodný, protože všechny klientské komponenty nabízí stejnou funkcionalitu. Pro uživatelské testy všech klientských komponent stačí navrhnout pouze jedno jediné zadání.

Uživatelské testování je jediný z procesů navržených pro zabezpečení QA ve službě ElateMe, které nebudou vykonávány s každou změnou zdrojového kódu. Tyto testy stačí vykonávat pouze při větších změnách uživatelského rozhraní nebo funkcionalit aplikace.

Kapitola 3

Continuous Integration

Continuous integration (CI) je postup usnadňující vývoj softwaru. K tomu se využívá specifických nástrojů a metod. CI usnadňuje integraci změn zdrojového kódu do hlavního repozitáře a zároveň minimalizuje riziko zavlečení chyby do funkcionality kódu v hlavním repozitáři. Případné chyby v kódu jsou detekovány brzy a proto mohou být jednoduše a levně opraveny. Další výhodou používání CI je úspora prostředků, jelikož proces je plně automatizovaný.

Tento postup má kořeny v agilní metodice vývoje softwaru. Agilní metodika vývoje vyžaduje velmi časté integrování změn do stávajícího kódu. Častá manuální integrace stojí nemalé množství prostředků, ale také vyžaduje opakované vykonávání stejného postupu. Pro minimalizaci těchto nákladů byla vynalezena CI. Jako jeden z prvních ji popsal a definoval Martin Fowler na svém blogu[9].

Výstupem správně implementované CI je funkční a vždy kompilovatelný kód v hlavní větvi sdíleného repozitáře, který je připraven k okamžitému nasazení do produkčního prostředí. Tato funkcionalita je velmi užitečná obzvlášť při větším počtu přispěvatelů do hlavního repozitáře, nebo při častých integracích změn. Continuous integration může být rozšířena o Continuous deployment nebo Continuous delivery pro automatizaci ještě větší části vývoje software.

Při testování multiplatformních aplikací je nutno vždy testovat kompatibilitu více komponent. Testovat se musí i kompatibilita komponent napříč všemi podporovanými verzemi a platformami. K tomuto účelu výborně poslouží automatizované testování realizované pomocí CI.

3.1 Požadavky Continuous integration

Většina postupů pro usnadnění vývoje má své požadavky, aby byly proveditelné. U CI tomu není jinak. Aby bylo možno implementovat CI, je nutno splnit několik základních požadavků, které vyplývají již z její definice. Splnění

těchto požadavků je mnohdy náročné, avšak jakmile jsou splněny, benefity CI značně převýší strasti spojené s její implementací.

3.1.1 Sdílený repozitář

Jelikož CI slouží k udržování zdrojového kódu ve sdíleném repozitáři, je hlavním požadavkem použití sdíleného repozitáře pro správu zdrojového kódu. Sdílené repozitáře jsou jednoduchými nástroji, které velmi efektivně usnadňují vývoj, proto jsou v současné době využívány u drtivé většiny projektů. Z tohoto důvodu nepředstavuje použití sdíleného repozitáře žádný problém. Typickým zástupcem těchto nástrojů je například Git[10].

3.1.2 Automatizovaný build

Protože je CI plně automatizovaný proces, musí být všechny jeho kroky proveditelné automatizovaně. Pro zjištění, zda lze provést build zdrojového kódu obsahujícího nové změny, je nutné tento build provést. Automatizovaného buildu lze lehce dosáhnout pomocí jednoduchého skriptu.

Buildování aplikace se většinou provádí na separátním serveru. Proto je k tomuto požadavku vhodné mít takový server k dispozici. Tento server se nazývá integrační server a slouží k buildování aplikace a jejímu následnému integračnímu testování.

3.1.3 Automatizované testy

Jistotu, že změny nenaruší možnost provést build zdrojového kódu, zajistí automatizovaný build. Pro zajištění funkcionality kódu pomocí CI slouží automatizované testy. Pro realizaci a usnadnění tohoto druhu testování existuje mnoho nástrojů.

Charakteristickou vlastností automatizovaných testů je, že probíhají bez dozoru živé osoby. O úspěchu či neúspěchu testu tedy rozhoduje pouze počítač. Tento fakt testování velmi stěžuje, jelikož zde chybí tzv. „lidský faktor“, který by mohl přihlédnout k okolnostem testu a vyhodnotit test objektivněji.

Pro účinné strojové testování funkcionality aplikace je velmi důležitý správný návrh testů. Tyto testy musí být jednoznačně strojově vyhodnotitelné a také musí pokrývat veškerou testovanou funkcionalitu zdrojového kódu. Návrh těchto testů je obtížný, avšak pro každou část funkcionality jej stačí vykonat pouze jednou.

3.1.4 Časté integrování změn

Tento požadavek není pro realizaci CI nezbytný, ale jeho dodržení zajistí využití maximálních výhod CI. Z důvodu automatizovanosti procesu tento požadavek nestojí žádné prostředky navíc. Při dodržení tohoto požadavku se případné chyby objeví brzy, kdy jsou ještě jednoduše opravitelné.

3.2 Proces Continuous Integration

Proces CI se vykoná při každém pokusu o začleňení změn do zdrojového kódu ve sdíleném repozitáři. Životní cyklus CI sestává ze čtyř základních částí: sestavení zdrojového kódu, build, testování a integrace změn. U každé této části je doporučena minimální funkcionalita, ale není výjimkou tuto funkcionalitu upravit, nebo přidat další části s novou funkcionalitou. Pokud nějaká část skončí neúspěchem, další části se již nevykonávají a změny se do hlavního repozitáře neintegrují. V tomto případě je vývojový tým informován o chybě. Chyby je nutno opravit a proces opakovat. Pro lepší představu je životní cyklus CI zachycen na obrázku 3.1.

1. **Sestavení zdrojového kódu**

   Jako první se sestaví celkový zdrojový kód aplikace. Toto sestavení proběhne tak, že do nejnovější verze zdrojového kódu se repozitář pokusí začlenit dané změny. Pokud při automatizovaném sestavování vznikne konflikt a počítač nebude schopen sestavit výsledný zdrojový kód zahrnující změny, proces je ukončen a je vyžadováno manuálního začlenění změn do kódu. Výstupem této části je zdrojový kód, který obsahuje zdrojový kód z hlavního repozitáře s integrovanými změnami. Tento krok vykoná sdílený repozitář a je vstupem pro další části CI.

2. **Build**

   Druhá část životního cyklu CI použije výsledný zdrojový kód z první části a pomocí připraveného buildovacího skriptu se pokusí o jeho build. Pokud build selže, proces je ukončen neúspěchem. V opačném případě je výstupem druhé části spuštěná aplikace. Tento krok vykonává nástroj pro využití CI.

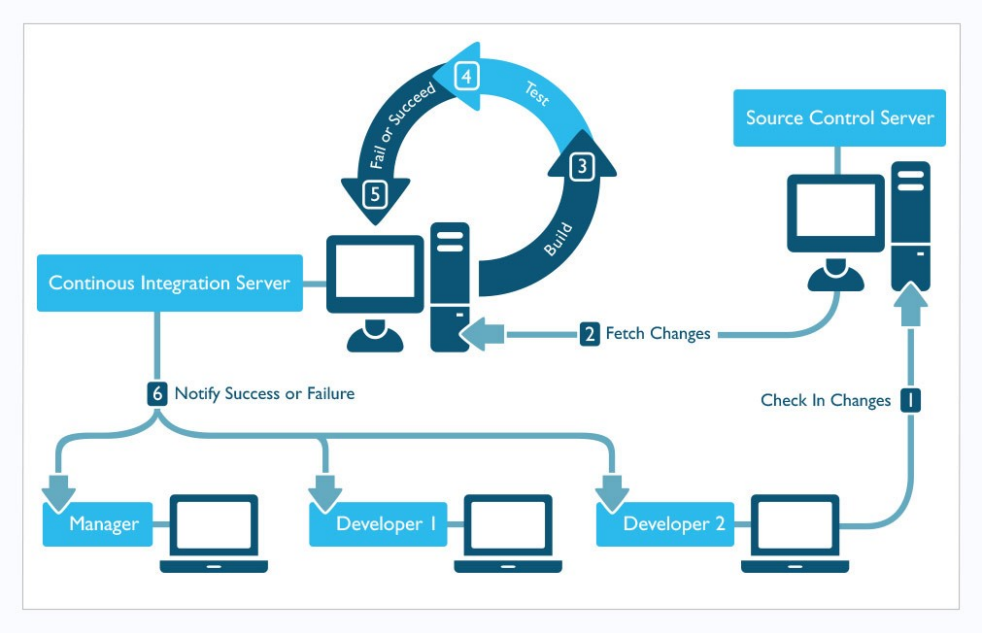
3. **Testování**

   Účelem této části je zkontrolovat funkcionalitu aplikace, která je výstupem předchozí části. Kontroly je dosaženo pomocí připravených automatizovaných testů, které jsou spuštěné vůči výstupu druhé části. Pokud testy odhalí chybu ve funkcionalitě, je tato část opět ukončena neúspěchem. Tento krok často bývá rozdělen na více částí, kde se v každé provádí jiný typ testování, za účelem co nejvyššího pokrytí funkcionality testy, vykonává jej nástroj pro využití CI a výstupem je informace, zda změny porušují některou z funkcionalit aplikace.

4. **Integrace změn**

   Pokud všechny předchozí části skončily úspěchem, má se za to, že změny jsou v pořádku a nenarušily současnou funkcionalitu aplikace. Zbývá je už jen zahrnout do hlavního repozitáře. To zajišťujě čtvrtá část životního cyklu CI. Sestavený zdrojový kód, který je výstupem první části, je

začleněn do hlavního repozitáře a je považován za nejnovější verzi zdrojového kódu aplikace. Tato část nemůže skončit neúspěchem a na jejím konci je vývojový tým informován o úspěchu integrace změn. Tento krok je vykonáván sdíleným repozitářem.



Obrázek 3.1: Životní cyklus Continuous integration [11]

3.3 Continuous delivery a Continuous deployment

CI slouží pro automatizované zajištění funkčnosti zdrojového kódu v hlavním repozitáři. Pro automatizované dodání (nasazení do produkce) provedených změn v aplikaci zákazníkovi slouží Continuous delivery a Continuous deployment. Tyto postupy slouží jako rozšíření procesu CI, tudíž vyžadují jeho implementaci. Oba postupy jsou si velmi podobné. Rozdíl je pouze v míře automatizace tohoto procesu, což má významný vliv na výhody a požadavky obou postupů. Rozdíly mezi Continuous delivery a Continuous deployment a jejich návaznost na CI jsou popsány například na webech digitalocean.com[12] nebo atlassian.com[13].

Jak Continuous delivery tak Continuous deployment spoléhají na správné provedení CI. Předpoklad, že zdrojový kód v hlavním repozitáři je funkční a připravený k nasazení do produkčního prostředí, je v těchto postupech zásadní. Výstupem těchto postupů je spuštěná aplikace v produkčním prostředí.

Z tohoto důvodu je nutno automatizovat nahrávání a spouštění aplikace na produkční prostředí. Tato funkcionalita se dá opět docílit pomocí skriptu. Do postupů obou procesů je vhodné implementovat další a rozsáhlejší testování, aby se co nejvíce zabránilo propagaci chyb do produkčního prostředí. Rozdílnost Continuos delivery a Continuous deployment a jejich návaznost na Continuous integration je znázorněna na obrázku 3.2.

3.3.1 Continuous delivery

U Continuous delivery se proces po úspěšném dokončení všech částí zastaví a vyčkává na manuální odsouhlasení nasazení aplikace do produkčního prostředí vývojářem. Tohle je jediná, ale zásadní odlišnost od Continuous deployment.

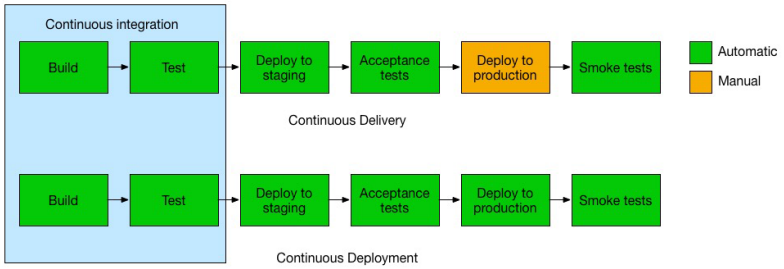
Díky vyčkávání na spuštění nahrávání na produkční server se zde otevírá možnost pro pokročilejší testování funkcionality aplikace. Častou praxí u Continuous delivery je nasadit aplikaci na předprodukční server, kde může být poskytnuta k dalšímu testování ať ze strany vyhotovitele či zákazníka. Tento proces navíc umožňuje aplikaci manuálně otestovat před spuštěním v produkčním prostředí.

3.3.2 Continuous deployment

U Continuous deployment probíhá celý proces automatizovaně bez jakéhokoliv zásahu vývojářů. To umožňuje velmi rychlé dodávání změn produktu zákazníkovi. Jelikož se nová funkcionalita dostává k uživatelům často a rychle, vývojový tým může průběžně analyzovat odezvy uživatelů a určovat směr dalšího vývoje aplikace.

Velký důraz je zde kladen na další automatizované testy, které musí pokrývat veškeré funkční i nefunkční požadavky, k zaručení maximální kvality vyvíjené aplikace. Největší nevýhodou tohoto procesu je fakt, že vývoj automatizovaných testů pro některé funkcionality je velmi nákladný a v některých případech i nemožný. Z tohoto důvodu bývají testy často nedostačující a je zde zvýšené riziko zavedení chyby do produkčního prostředí.

Prestože rychlá odezva od uživatelů je velkým přínosem, vyvstává otázka, zda je vhodné nechat počítači naprostou kontrolu nad tímto procesem. Při nedostatečné konfiguraci procesu nebo návrhu a implementaci testů, by mohlo být velmi rychle dosaženo značných ztrát. Rozhodnutí, zda je výhodné podstoupit toto riziko, vždy záleží na manažerovi. Já osobně si myslím, že částečná kontrola člověkem je mnohem výhodnější, protože může zabránit potenciálním katastrofám.



Obrázek 3.2: Continuous delivery, Continuous deployment a jejich naváznost na Continuous integration[13]

3.4 Nástroje použité pro Continuous integration v ElateMe

Nástrojů pro realizaci CI existuje mnoho. Z důvodu, že projekt již používal pro správu ropozitáře nástroj GitLab, a na přání zadavatele byli použity nástroje GitLab CI/CD a Docker. V této sekci je stručně popíšu.

3.4.1 GitLab

GitLab[14] je nástroj pro komplexní správu Gitového repozitáře. Může být nainstalován na vlastním serveru, nebo lze využít hostovaných repozitářů na webu GitLab.com. Mimo správy Gitového repozitáře nabízí GitLab i mnoho dalších služeb sloužících ke zjednodušení vývoje software. Jednou z těchto služeb je i GitLab CI/CD, která slouží pro zajištění CI. K použití této služby je vyžadována aplikace GitLab runner.

GitLab runner je aplikace, která vykonává automatizované kroky CI, jako jsou například build, testování, deploy, atd. Runner je doporučeno mít nainstalován na jiném serveru, než tam, kde je nainstalován Gitlab. Pomocí koordinačního API Gitlab spouští skrze runner na serveru definované procesy CI, který každý proces vzkoná, vyhodnotí a výsledky odešle zpátky Gitlabu.

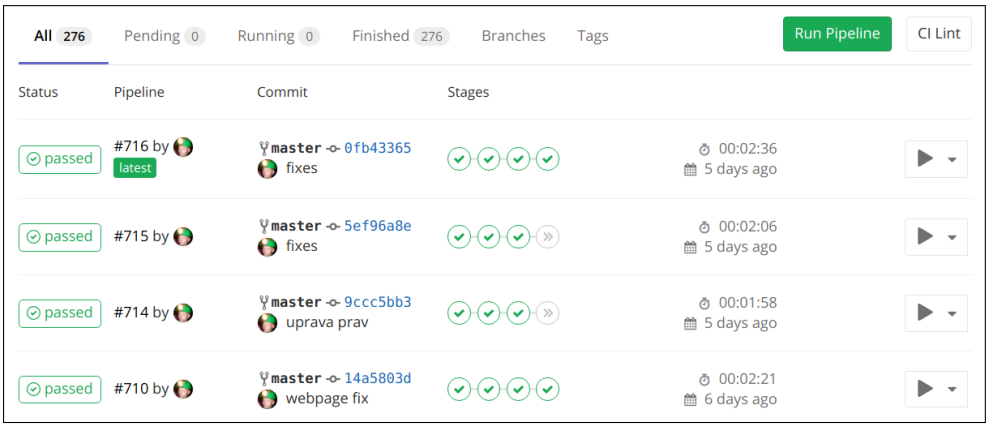
K použití služby pro zajištění CI poskytované GitLabem, stačí pouze přidat soubor gitlab-ci.yml do kořenové složky repozitáře, nainstalovat Gitlab runner a nakonfigurovat projekt, aby runner využíval. Poté každý commit do

repozitáře spustí proces CI. Jednotlivé spuštění procesu CI GitLab se nazývá pipeline. Každá pipeline se může skládat z více částí, ty se nazývají stage.

Všechny stage, které se při každé pipeline spustí, jsou definovány a nakonfigurovány v textovém souboru gitlab-ci.yml. Konfigurace v tomto souboru musí být ve formátu YAML. Jelikož je gitlab-ci.yml verzován zároveň se zdrojovým kódem, je možno jej upravit na míru každé verzi zdrojového kódu.

Jakmile jsou postupy CI nastaveny, GitLab je provádí automatizovaně při každém pokusu o integraci změny do zdrojového kódu. Evidovány jsou všechny spuštěné pipelines a informace o průběhu jejich stages. Zobrazované informace o proběhlých pipeline si čtenář může prohlédnout na obrázku 3.3.

Pro usnadnění přípravy prostředí na integračním serveru, kde je spuštěn GitLab runner, je vhodné použít nástroj docker. Gitlab poskytuje službu GitLab registry, která slouží k ukládání dockerových image a jejich následnému možnému využití v procesech CI.



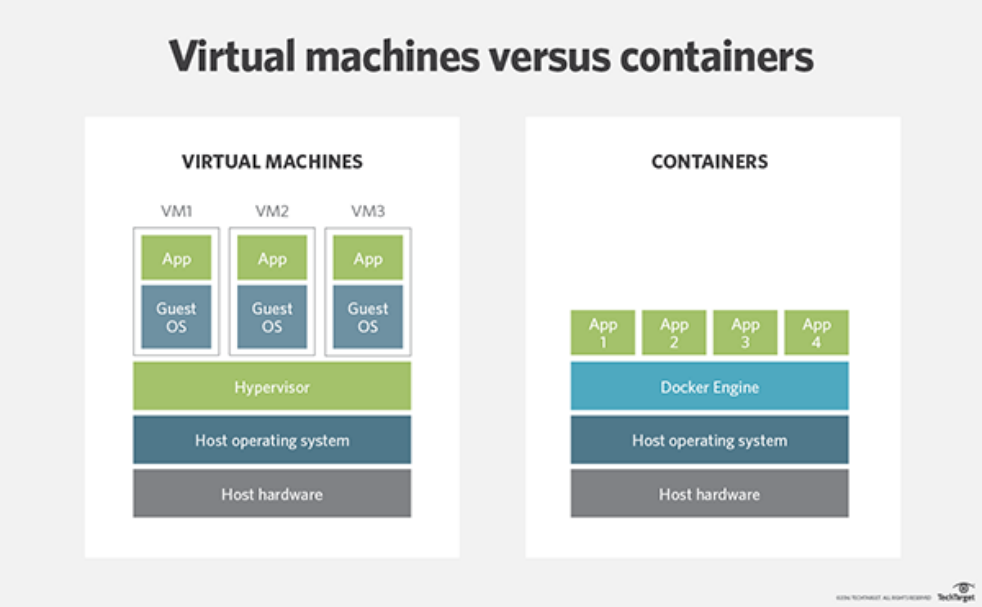
Obrázek 3.3: Přehled vykonaných pipelines poskytovaný GitLabem

3.4.2 Docker

Docker[15] je moderní nástroj pro kontejnerizaci software. Poskytuje platformu s jednotným prostředím pro běh aplikací, které jsou uzavřené do tzv. kontejnerů. Kontejnery slouží k izolované simulaci prostředí pro běh aplikací. Každý kontejner obsahuje danou aplikaci a potřebné nástroje k jejímu spuštění. Práce s kontejnery je jednoduchá a může být automatizována.

Funkcionalita kontejnerizace je velmi podobná virtualizaci s tím rozdílem, že kontejnery neobsahují vlastní operační systém, ale mají přístup ke zdrojům operačního systému, na kterém jsou hostovány, a propagují je dovnitř sebe. Z tohoto důvodu není nutné hostovat více operačních systémů a stačí pouze jeden. To šetří výkon počítače a umožňuje na stejném stroji spustit několikanásobně více kontejnerů než virtuálních strojů, které vykonávají tu

samou činnost. Porovnání virtuálních strojů a kontejnerů čtenář nalezne na obrázku 3.4



Obrázek 3.4: Porovnání virtuálních prostředí a kontejnerů[16]

Docker nabízí online úložiště Docker Hub[17], které obsahuje spousty předpřipravených image různých prostředí, které se poté jen přizpůsobí danému využití za pomoci dockerfile. To je další z mnoha výhod Dockeru, jelikož není nutno vytvářet a konfigurovat prostředí od začátku. Dockerfile je soubor, který obsahuje šablonu a postup pro automatizované vytvoření image kontejneru.

Z dockerfile Docker vytvoří (buildne) spustitelný image kontejneru. Kontejner je spuštěný image, který může být spuštěn i vícekrát. Pro vytvoření určitého kontejneru tedy stačí pouze textový soubor, který zabírá zanedbatelné množství místa na disku. Velká výhoda vytvořených image je ta, že jsou lehce přenosné, protože v sobě nesou veškerá data a nástroje pro jejich bezproblémové spuštění na docker platformě.

Vlastnosti kontejnerizace přináší obrovské usnadnění automatizace procesu CI. Pro každou aplikaci stačí vytvořit jeden dockerfile pro přípravu prostředí a instalaci aplikace, pomocí kterého se automatizovaně vytváří image kontejnerů. Tyto image lze poté lehce přenášet a spouštět mezi servery a provádět na nich testování aplikace. Navíc lze pomocí image skladovat všechny verze aplikací, které jsou poté lehce přístupné a použitelné.

3.5 Continuous integration pro službu ElateMe

Cílem CI u aplikace ElateMe je automatizovat co možná největší část procesu vývoje všech komponent. Úspěšné implementování CI může usnadnit mnoho nepříjemností spojených s vývojem aplikace a navíc má potenciál ušetřit značné množství času a nákladů. Aplikace ElateMe se skládá za 4 komponent, které jsou vyvíjeny zaráz. Je nutné testovat kompatibilitu všech komponent dohromady (zejména jednotlivých klientských komponent se serverovou komponentou), aby bylo zajištěno kvality aplikace jako celku. Navíc je požadavek testovat kompatibilitu těchto komponent i napříč verzemi. Manuální testování při každé změně by stálo mnoho času, o financích ani nemluvě.

Jelikož je k uchovávání a verzování zdrojového kódu všech komponent aplikace ElateMe použit GitLab, je vhodné využít jeho nástrojů pro implementaci CI. Pro usnadnění automatizovaných procesů pro Continuous integration je využit nástroj Docker. Kostra se základními částmi CI pro serverovou komponentu, která umožňuje build komponenty, a její manuální nasazení do produkčního a předprodukčního prostředí, již byla úspěšně implementována. Při psaní této práce jsem doimplementoval další část, která připravila prostředí a zajistila automatizované vykonání testů API.

3.5.1 Návrh testovacího prostředí

Jak již bylo zmíněno, pro vykonávání CI pomocí nástroje Gitlab CI/CD je nutno mít nainstalovaný nástroj GitLab Runner. Pro vykonávání integračních skriptů slouží integrační server, na kterém jsou mimo jiné nainstalovány GitLab Runner a Docker. To umožňuje ve skriptech CI využívat výhod a funkcionalit Dockeru.

Dockerem disponuje i předprodukční prostředí, na které budou nahrávany aplikace za účelem akceptačního a závěrečného manuálního testování před nasazením do produkčního prostředí.

Pro usnadnění procesu CI a plné využití Dockerových kontejnerů, je využito služby Gitlab registry. V tomto úložišti jsou uloženy předpřipravené image, potřebné pro vykonávání procesů CI:

- **Image obsahující ssh klíče**

  Kontejner vzniklý spuštěním tohoto image je vhodný pro přenos dat mezi servery. Jelikož jsou klíče v image již obsaženy není potřeba je nikde uchovávat a snižuje se tím riziko jejich odcizení neautorizovanou osobou.

- **Image obsahující testovací nástroje a testy**

  Pomocí těchto image lze jednoduše automatizovaně spouštět testy vůči kontejnerům obsahujícím běžící instance potřebných verzí aplikace.

- **Image obsahující podporované verze platforem**

  V těchto image lze spouštět aplikace, vůči kterým lze spouštět testy které ověří správnost chování aplikací skrze verzemi platforem.

- **Další image, které se postupem času ukážou jako užitečné pro proces CI**

Pro účely testování změn budou při každém běhu vytvářeny nové docker image obsahující dané verze komponent. Vůči těmto kontejnerům budou poté spouštěny automatizované testy. Při neúspěchu testů budou image jednoduše smazány. Při úspěchu budou image přeneseny a spuštěny na předprodukčním serveru pro účely dalšího testování. Po úspěšném absolvování veškerého testování budou tyto image uloženy v GitLab registry a posléze mohou být přímo nasazeny do produkčního prostředí.

3.5.2 Proces CI

Pro vývoj komponent ElateMe byl zvolen postup Continuous integration rozšířen o Continuous deployment. Jelikož většina testů pro komponenty ještě není vyvinuta, nebo není ve stavu, ve kterém by mohla výrazně minimalizovat rizika chyb, je nutno do procesu CI začlenit i manuální testování aplikace. Tímto se utlumí projevy nedostatků automatizovaných testů. Jakmile budou vyvinuty plnohodnotné automatizované testy, je možné vynechat nebo automatizovat některé manuální části procesu CI. I přesto je v procesu vhodné nechat alespoň jednu manuální čast, pro kontrolovaný průběh nasazení komponent do produkce.

Navržené procesy CI se liší dle větve do které jsou nahrány změny. Tyto procesy se spustí při každém nahrátí změn do repozitáře. Procesy jsou navrženy tak, aby usnadnily současný postup vývoje služby ElateMe. V tomto postupu vývojáři nahrávají změny do svých větví. Tyto větve jsou začleňeny do hlavní větve až po kontrole zdrojového kódu seniorním programátorem. Proto je proces CI rozdílný pro vývojové větve a hlavní větev.

3.5.2.1 Části procesu CI pro vývojové větve repozitáře

1. **Build kontejneru s danou verzí komponenty**
2. **Unit testy**
3. **Akceptační testy**
4. **Revize zdrojového kódu seniorním programátorem**

   Dokončení celého procesu CI je nutno manuálně odsouhlasit po provedení revize kódu. Zde je možnost provést případné manuální testování. V případě uznání za vhodné, je provedeno manuální začlenění (merge)

změn do hlavní větve repozitáře, kde je následně spuštěn proces CI pro hlavní větev.

Do vývojových větví jsou nahrávány změny bez jakékoliv předchozí revize, tudíž mají největší pravděpodobnost výskytu chyb. Z toho důvodu jsou provedeny nejdříve automatizované testy, aby tyto chyby odhalily a nedocházelo k plýtvání času seniorního programátora. Teprve až po uspění ve všech testech je zdrojový kód kontrolován. Dokončení tohoto procesu je nutné manuálně potvrdit, aby bylo možné provést případné manuální testování, které je kvůli úspoře zdrojů v této části doporučeno provádět pouze v podezření na porušení funkcionality, nebo při nedostatečných automatizovaných testech.

3.5.2.2 Části procesu CI pro hlavní větev repozitáře

1. **Build kontejneru s danou verzí komponenty**
2. **Unit testy**
3. **Akceptační testy**
4. **Nasazení aplikace do předprodukčního prostředí**
5. **Testy kompatibility verzí aplikace napříč verzemi platforem**

   Tato část je volitelná, protože je často časově náročná a prodlužovala by dobu oprav kritických chyb, které musí být co nejrychleji nasazeny do produkčního prostředí.
6. **Výkonnostní testování**

   Tento krok je vykonáván pouze u serverové komponenty.
7. **Manuální přijmutí nové verze**

   Zde je prostor pro manuální nebo uživatelské testování funkcionality. Při přijmutí verze dojde k označení daného dockerového Image a jeho uložení do GitLab registry. Následně bude daná verze aplikace automatizavaně nasazena do produkčního prostředí.

Spuštění procesu CI pro hlavní větev repozitáře proběhne výhradně po manuálním začlenění změn z vývojové větve pověřenou osobou. První tři části procesu CI pro hlavní větev jsou stejné jako pro vývojové větve, protože není vyloučeno, že hlavní větev obsahuje změny zdrojového kódu nezahrnuté ve vývojové větvi. Následuje nasazení do předprodukčního prostředí a pokročilejší testy, které mají za úkol ověřit připravenost aplikace na spuštění v produkčním prostředí. Na konci je opět vyžadováno manuální dokončení procesu, aby bylo umožněno manuální testování. To je v této fázi silně doporučeno, obzvláště testování automatizovatelně netestovaných funkcionalit jako jsou donace. Rovněž

by měli být provedeny manuální testy funkcionalit, které nejsou pokryty automatizovanými testy. Po manuálním dokončení této části jsou automatizovaně provedeny veškeré činnosti pro vydání nové verze.

Kapitola 4

Testování API

Z analýzy vyplynulo, že funkčnost API na serverové komponentě služby ElateMe je pro bezproblémový chod aplikace zásadní. Jakákoliv chyba, která se projeví na této komponentě, může mít kritický vliv na popularitu této služby. Je vyžadováno, aby automatizované testy API byly co nejdůkladnější a pokryly pokud možno veškerý rozsah funkcionality API.

Kvalitu API u aplikací, kde je předpokládán počet uživatelů v řádech tisíců a vyšší (ElateMe patří mezi tyto aplikace), neurčuje pouze míra chybovosti ve funkcionalitě. Důležitým ukazatelem kvality jsou také nefunkční parametry API. Mezi tyto parametry se řadí zejména průměrná délka odezvy, dostupnost aplikace a další. Většinu těchto parametrů lze otestovat výkonnostním testováním.

Značným ztížením automatizování testů ElateMe API je fakt, že k úspěšnému přihlášení uživatele je potřeba použít služeb třetí strany a to Facebook API[18]. Velkou výhodou Facebook API je, že poskytuje velmi kvalitní a účinnou podporu pro automatizované testování aplikací, které jej využívají. Každá aplikace, která chce využívat benefitů Facebook API, musí mít vytvořený účet, který je s ní svázán. Skrze tento účet může aplikace využívat jednotné přihlášení uživatelů Facebooku a také získá přístup k určitým uživatelským datům Facebooku.

K účelům testování umožňuje zaregistrovaná facebooková aplikace vytvoření fiktivních testovacích uživatelů pro danou aplikaci, kteří jsou izolovaní od reálných uživatelů. Testovací uživatelé nikdy nepřijdou do styku s reálnými uživateli, tudíž nemůže žádným způsobem dojít k narušení chodu aplikace nebo ovlivnění reálných uživatelů. Testovací uživatele je možné upravit dle potřeb testování aplikace, přátelství je možné navazovat pouze mezi testovacími uživateli. K úspěšnému přihlášení uživatele do ElateMe je potřeba získat přístupový token, který vydává právě Facebook API.

4.1 Testování funkcionality

Testování funkcionality API je nejzákladnějším způsobem jak změřit jeho kvalitu. Součástí této práce je návrh i implementace těchto testů. Tyto testy slouží jako automatizované akceptační testy pro serverovou komponentu. Tato sekce obsahuje detailní popis testování funkcionality API na serverové komponentě ElateMe. Pro implementaci testů byl použit nástroj Apache JMeter.

4.1.1 Automatizovaně netestovatelné funkcionality

Skutečnost, že API na serverové komponentě poskytuje data určená k dalšímu strojovému zpracování, velmi usnadňuje její automatizované testování. Navzdory tomu existují funkcionality, které nelze strojově otestovat. Mezi takovéto funkcionality patří zejména ty, u kterých je potřeba smysluplné lidské interakce, nebo u nich nelze jistě předpovědět, kdy budou dokončeny. I přesto, že při možnostech dnešní techniky je možné na mnoho takovýchto funkcionalit vyvinout automatizované testy, manuální testování v těchto případech je stále mnohonásobně levnější a výhodnější.

ElateMe nabízí jednu automatizovaně netestovatelnou funkcionalitu, a to zrovna tu hlavní. Jelikož donace pracují s finančními prostředky a k jejich zpracování je potřeba využít funkcionalitu třetí strany, je tato funkcionalita velmi obtížně strojově testovatelná. Náročná implementace testů, které využívají funkcionalit dvou různých aplikací, není hlavním problémem při automatizaci těchto testů. Rozhodnutí o úspěchu nebo neúspěchu testu může být vykonáno až po provedení transakce bankou, což obvykle trvá řádově několik dní. V automatizovaném testování (obzvláště v tomto konkrétním případě) je takto dlouhé vyhodnocení testů nežádoucí.

Z výše uvedených důvodů se funkcionalita týkající se donací netestuje automatizovaně. K testování této funkcionality budou použity manuální testy v předprodukčním (a případně produkčním) prostředí.

4.1.2 Návrh testů

Jak již bylo zmíněno, od testů funkcionality API je vyžadováno důkladně pokrýt co největší rozsah jeho funkcionality. Správně navrhnuté a implementované testy mohou včas odhalit chyby a nedostatky nejen v současné ale i v budoucích verzích API.

Po konzultaci se zadavatelem jsem se rozhodl, že pro aktuální stav aplikace je vhodné, aby každý test funkcionality simuloval jedno přihlášení několika uživatelů. Při tomto běhu budou otestovány všechny přístupové body API. Zároveň je od testů vyžadováno, aby šli spustit vícekrát paralelně, což umožňuje simulovat běh aplikace při zatížení mnoha uživateli. Z tohto důvodu mohou být tyto testy využity i k orientačnímu testování výkonnosti API, protože výkonnostní testy ještě nebyly implementovány.

Od testů se očekává, že budou spouštěny v dockerovém kontejneru vůči dalšímu dockerovému kontejneru, který bude obsahovat danou verzi serverové komponenty k otestování. Proto je vyžadováno, aby byly flexibilní a bylo možné nakonfigurovat jejich parametry, se kterými budou spouštěny.

Jelikož uživatelé v ElateMe mohou vystupovat pod několika rolemi, je u testů vyžadováno nejen ověření správné funkčnosti API, ale také kontrola autorizace a dostupnosti dat. Ukázkový příklad je viditelnost přání. Přání si může prohlédout přihlášený uživatel, který jej založil, nebo je v přátelství s uživatelem, kterému je přání určeno a nebo sám uživatel pro kterého je dárek určen v případě, že je přání veřejné. Nepřihlášení uživatelé nebo uživatelé, kteří nenavázali facebookové přátelství s vlastníkem přání, nesmí mít přístup k údajům přání, či uživatelským údajům vlastníka přání. Všechny tyto varianty přístupu musí být testovány pro každou funkcionalitu poskytovanou API na serverové komponentě. Tímto se výrazně minimalizují rizika zapříčiněná vznikem bezpečnostních děr.

V úvodu testu je nutno vytvořit a vhodně nakonfigurovat nové testovací uživatele pomocí Facebook API a získat jejich přístupové tokeny. Pro provedení testu jsem zvolil čtyři testovací uživatele. Jelikož funkcionalita aplikace významně rozlišuje, zda jsou uživatelé na Facebooku přátelé, při vytváření testovacích uživatelů se vzájemně uzavře facebookové přátelství mezi třemi testovacími uživateli. Čtvrtý testovací uživatel nenavazuje žádné přátelství.

Jakmile jsou testovací uživatelé nakonfigurováni, začíná testování funkcionality. Testování funkcionality pokrývá veškeré uzly API s výjimkou těch, které jsou automatizovaně netestovatelné. Jako první se testují funkcionality, které nevyžadují žádné předchozí kroky. Mezi tyto funkcionality patří zejména přihlášení uživatelů do aplikace a kontrola údajů převzatých od Facebook API.

Následující testy slouží k ověření funkcionality vytváření produktů. Na tyto testy navazují další testy, které ověřují funkcionalitu využívající vytvořených produktů. Mezi tyto testy patří například vytvoření přání a následně jeho změny. Samozřejmostí jsou i pokusy o neautorizované zásahy, které by mělo funkční API zamítnout. Při těchto testech je nutné využívat jak uživatele, kteří jsou navzájem přátelé, tak i uživatele, který nemá uzavřené žádné přátelství.

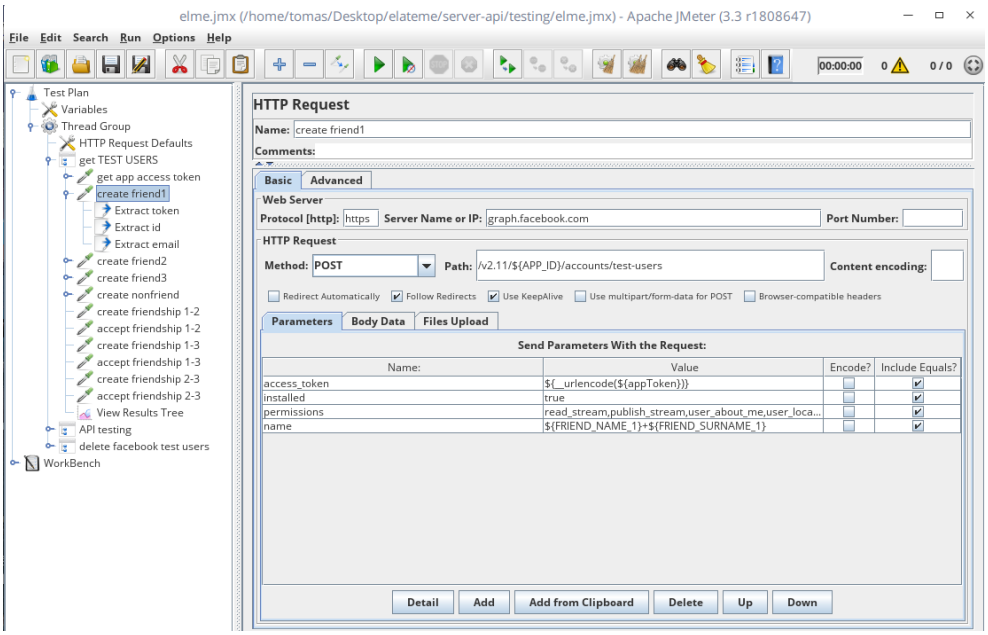
Závěrečné testy slouží k ověření funkcionality odstranění vytvořených produktů. Účelem těchto testů je nejen ověřit funkcionalitu mazání produktů, ale hlavně otestovat zamezení neautorizovaným mazáním. Úplně poslední testy ověřují odhlášení z aplikace. Po skončení testování funkcionality se za pomocí Facebook API smažou testovací uživatelé, kteří byli pro tyto testy vytvořeni.

4.1.3 Apache JMeter

Apache JMeter[19] je opensource nástroj sloužící k automatizovanému testování funkcionality a výkonu software. Mezi podporované typy software pro testování patří i RESTové služby. Nejen proto je Apache JMeter vhodný nástroj pro implementaci automatizovaných testů ElateMe API.

JMeter je napsán v Javě, proto jej lze spustit na všech platformách, které podporují JVM (Java virtual machine). V současné době je JVM podporováno téměř všemi používanými platformami, proto je JMeter snadno dostupný pro téměř všechny platformy.

Veškerá implementace testů probíhá v grafickém uživatelském rozhraní, které je vyobrazeno na obrázku 4.1. Mnoho funkcionalit a způsobů testů zajišťuje co největší flexibilitu při navrhování testů. Vyhodnocení testů zajišťují předpřipravené lehce konfigurovatelné funkcionality. Pokud tyto funkcionality neumožňují vyhodnocení vhodným způsobem, lze pro vyhodnocení využít BeanShell skripty, které umožňují provést téměř jakékoliv automatizované vyhodnocení. Výsledky testů mohou poté být prezentovány přímo v grafickém prostředí, nebo mohou být uloženy v mnoha formátech.



Obrázek 4.1: Grafické uživatelské rozhraní Apache JMeter

Jelikož od vydání první verze uplynulo několik let, má JMeter početnou komunitu uživatelů. Z tohoto důvodu je velmi snadné dohledat informace o možných problémech. Pro JMeter je vyvinuto mnoho pluginů, kterými lze rozšířit základní funkcionalitu. Těmito pluginy lze docílit například ještě větší flexibility při vykonávání testů a jejich vyhodnocení, nebo také upravit způsob prezentování výsledků testů.

Jmeter umožňuje spuštění testů za použití příkazové řádky. Nejenže to značně ulehčuje automatizaci testů, ale navíc je tímto způsobem dosaženo přesnějších výsledků výkonnostních testů, jelikož grafické rozhraní snižuje výkon stroje, na kterém jsou testy spuštěny.

Vytvořené testy jsou uloženy do textového souboru ve formátu XML, díky tomu lze testy jednoduše verzovat zároveň s testovaným softwarem. Díky možnosti spouštění testů pomocí příkazové řádky a jejich snadné verzovatelnosti je Jmeter vhodný pro použití v Continuous integration.

4.1.4 Implementace testů

Automatizované testy funkcionality API byly implementovány dle výše uvedeného návrhu. Při jejich realizaci se objevilo několik nedostatků API. I přes značné obtížnosti v průběhu implementace testů se testy podařilo úspěšně implementovat. Jejich shrnutí naleznete na konci této sekce v tabulce pokrytí 4.1.

Jako první neodstatek se ukázala nedostatečná dokumentace API. Dokumentace je pouze automaticky generovaná na základě implementovaných funkcionalit. Tato dokumentace neobsahuje ani správné požadavky na obsah HTTP požadavku, ani příklad správné odpovědi a možných hodnot. Tento přístup značně stěžuje nejen testování API, ale i vývoj. Jedinou výhodou stávající dokumentace je možnost si s její pomocí ověřit funkcionalitu API.

Implementace testů byla z důvodu špatné dokumentace velmi zdlouhavá. Často jsem musel zkoušet více možných variant abych „uhodl“ správnou formu HTTP požadavku. Ještě náročnější bylo vyhodnotit odpovědi API na implementované požadavky. Jelikož dokumentace nepopisuje žádné správné formáty ani krajní případy odpovědí API, musel jsem často konzultovat se zadavatelem, vývojáři a testovacím týmem správnost mnou implementovaných testů. Tento způsob implementace byl několikanásobně zdlouhavější, než je obvyklé. Vytvoření dostatečné dokumentace by ušetřilo čas nejen testerům, ale i vývojářům.

Další obtížností při implementaci testů byla skutečnost, že se API neustále vyvíjelo a upravovalo. Se současnou nedostačující dokumentací tato skutečnost nejvíce zpomalovala vývoj testů. Testy se musely často předělávat, jelikož funkcionalta, pro kterou byly vytvořeny, byla změněna nebo úplně odstraněna. Byly nutné časté konzultace se zadavatelem a vývojovým týmem, protože nikde nebyl popsán konečný stav a funkcionalita API. Z tohoto důvodu nebylo možné implementovat testy na ještě neimplementované funkcionality. Vzhledem k častým změnám v API je nutné testy často upravovat, aby byly kompatibilní se současným stavem API.

I přes všechny obtížnosti byly testy implementovány, a pokrývají všechny implementované automatizovaně testovatelné funkcionality ElateMe API. Jejich shrnutí je zaznamenáno v následující tabulce pokrytí 4.1.

Tabulka 4.1: Tabulka pokrytí testy

API uzel	Pokryto	poznámka
account	Ano	
account/<id>	Ano	
account/restricted/<id>	Ano	
account/auth/login	Ano	
account/auth/logout	Ano	
account/firends	Ano	
account/friends/user/<userId>	Ano	
account/friends/synchronize	Ano	
account/friends/birthdays	Ano	
account/friends/groups	Ano	Vytváření skupiny vrací HTTP status kód 500
account/friends/groups/<id>	Ne	Netestovatelné, jelikož nefunguje vytváření skupin
account/friends/groups/colors	Ano	
wishes	Ano	
wishes/webpage	Ne	Neimplementovaná funkcionalita
wishes/contributed	Ne	Nejsou implementovány donace
wishes/user/<userId>	Ano	Nezobrazují se přání od přítele
wishes/<wishId>	Ano	HTTP metoda PUT vrací kód 500
wishes/upload/image	Ne	Neimplementováná funkcionalita
wishes/restricted/wish	Ano	
wishes/suggested	Ano	Na dokumentaci vrací HTTP kód 500
wishes/<wishId>/comments	Ano	Při posílání pomocí POST parametrů vrací kód 500, nicméně na dokumentaci funguje jelikož data posílá jinak než pomocí POST parametrů
wishes/<wishId>/comments	Ano	Nevyhodnotitelné testy, protože nejdou vytvářet komentáře
donations	Ne	Neimplementovaná funkcionalita, část je netestovatelná
donations/wish/<wishId>	Ne	Neimplementovaná funkcionalita

donations/cardpayment	Ne	Neimplementovaná funkcionalita
feed	Ano	
feed/hide/wish/<wishId>	Ano	Chová se jakoby uspělo, ale přání je stále viditelné
feed/hide/user/<userId>	Ano	
notifications	Ano	Bohužel jen omezený rozsah, protože nelze přidávat komentáře
notifications/popups	Ne	Neimplementovaná funckcionalita
notifications/count	Ne	Neimplementovaná funckcionalita

4.1.5 Nalezené chyby

Jelikož se testy začaly vyvíjet později, než serverová komponenta, bylo při jejich implementaci nalezeno několik skrytých chyb. Jelikož tyto chyby neporušovaly implementované funkcionality klientských komponent, neodhalilo je žádné jiné do té doby provedené testování.

Nejkritičtější nalezená chyba byla v synchronizování facebookových přátel. Pokud bylo na straně Facebooku navázáno nebo zrušeno přátelství, občas se tato změna neprojevila v aplikaci ElateMe. To znamenalo, že uživatel nemohl sledovat přání nebo účty nově získaných přátel. Při manuálním testování se tato chyba neprojevila, nebo nebyla odhalena. Nicméně při automatizovaných testech se chybu povedlo několikrát vyvolat, a co nejpřesněji popsat okolnosti, za kterých se projevuje. Chybu se později podařilo potvrdit i manuálním testováním a následně byla opravena.

Další objevené chyby se převážně týkaly nefunkčnosti některých funkcionalit. Převážně šlo o chyby, kdy API místo očekávaného HTTP stavového kódu a odpovědi, odpovídalo kódem 500 – Internal server error s podrobným výpisem, kde se chyba objevila.

Vyhodnocení objevených chyb ukázalo, že mnoho částí funkcionalit není funkčních, tudíž aktuální verze serverové komponenty momentálně není schopná provozu v produkčním prostředí.

4.2 Návrh výkonnostních testů

Výkonostní testy slouží pro měření nefunkčních požadavků na aplikaci. Nedostatečný výkon může být zapříčiněn převážně zvolením nevhodných algorytmů pro obsluhu dotazů na API nebo nedostatečné výpočetní síly serveru, na kterém API běží. Další nechtěné dopady na výkonnost API může mít například nastavení serveru a nebo nadměrné množství dat v databázi. Z tohoto důvodu

je nutné stanovit si požadované hodnoty na výkon serverovou komponentu a kontrolovat, zda je těchto hodnot dosaženo. Toto testování je prevencí pro rizika spjatá se snížením výkonu serverové komponenty.

Služba ElateMe cílí mimo tuzemského trhu i na ty zahraniční, proto je očekáván velký počet uživatelů využívajících tuto službu. Z tohoto důvodu je vyžadováno mít dostatečně výkonnou serverovou komponentu, která je schopna obsloužit dotazy na API v „rozumném“ čase. „Rozumný“ čas pro průměrnou odezvu API byl stanoven na 500 ms. Zároveň je požadováno aby tento výkon byl udržen i při plném zatížení aplikace, který byl u služby ElateMe určen pro začátek na 1000 uživatelů v jeden okamžik. Tento počet bude postupem času růst zároveň s počtem uživatelů ElateMe. Mimo jiné je vhodné mít informace o maximálních možnostech zatížení serverové komponenty, k jejich změření jsou využívány tzv. stress testy.

Jelikož tyto testy mohou zahltit, nebo dokonce způsobit selhání serverové komponenty, není vhodné je vykonávat vůči produkčnímu prostředí. Z tohoto důvodu je nutno mít k dispozici prostředí, kde je možné otestovat výkon serverové komponenty. Toto prostředí by mělo být stejné jako produkční prostředí, z ekonomických důvodů však je nevýhodné provozovat dvě taková prostředí, proto předprodukční prostředí disponuje nižším výkonem než produkční prostředí. Toto je značné znesnadnění pro měření výkonnosti, jelikož testy je nutno provádět pro nižší parametry, které zvládne předprodukční server obsloužit, a poté tyto výsledky přepočítat a vztáhnout na serverovou komponentu.

Pro měření požadovaného výkonu serverové komponenty je zapotřebí pouze jeden testovací scénář, který bude spouštěn vícekrát pro měření různých parametrů. Aby bylo docíleno co nejpřesnější simulace postupu uživatelů při využívání API, je nutno odhadnout četnost různých druhů požadavků na API. Je zřejmé, že např. přání budou mnohem častěji zobrazována než vytvářena nebo mazána. Pro usnadnění budou v testovacím scénáři využiti 3 testovací uživatelé. Ve scénáři se poté budou vykonávat všechny dotazy na poskytovanou funkcionalitu, avšak s rozdílnou četností. Počet uživatelů, kteří budou vykonávat určité požadavky naleznete v tabulce 4.2.

Tabulka 4.2: Tabulka odhadu četnosti dotazů na API.
Četnost 1/3 znamená, že tyto požadavky bude vykonávat pouze jeden uživatel ze tří, tudíž budou 3 krát méně časté než požadavky s četností 3/3.

Četnost u uživatelů	Požadavky
3/3	Veškeré GET požadavky
2/3	Požadavky na přidávání/mazání komentářů
1/3	Ostatní požadavky

Jako první budou vykonány výkonnostní testy pro ověření, zda daná verze serverové komponenty splňuje požadavky na průměrnou odezvu při daném za-

tížení uživateli. Výsledky těchto testů je nutno zaznamenat pro pozdější analýzu seniorním programátorem. Je nutno uchovávat průměrnou odezvu všech typů požadavků, což umožňuje detekci případných výkonnostních nedostatků v implementaci daného požadavku. Výsledky je nutno porovnávat s výsledky výkonnostních testů předchozích verzí, aby bylo možno odhalit případné snížení výkonu způsobené změny v implementaci.

Následují stress testy. V těchto testech bude testováno maximální možné zatížení serverové komponenty. Tyto testy budou sestávat z opakovaných spouštění navržených testovacích scénářů. Tyto testy začnou na definovaném počtu uživatelů pro výkonnostní testy a s každým během se tento počet se zvýší o počáteční hodnotu počtu uživatelů. Tyto testy budou opakovány do selhání serverové komponenty v důsledku přetížení, nebo dokud průměrná odezva na dotaz nepřesáhne hodnotu 2000 ms. Úkolem těchto testů je zjistit maximální možné zatížení serverové komponenty, aby se s případným nárůstem uživatelů mohlo včas začít se zvyšováním výkonnosti. Dalšími daty, které je nutno zaznamenat z těchto testů, je maximální počet uživatelů, pro které průměrná odezva stále dosahuje požadovaných hodnot.

Závěr

Hlavním cílem této práce práce bylo navrhnout a částečně implementovat procesy QA pro multiplatformní aplikaci ElateMe. Před návrhem těchto procesů bylo nutno provést analýzu změnových požadavků, která ukázala že tyto požadavky s sebou přináší různé druhy rizik, které mají navíc různou váhu dopadů na jednotlivé komponenty aplikace. I přesto jdou příčiny všech rizik shrnout jako pravděpodobnosti výskytu chyb a téměř všechny dopady ústí v pokles uživatelské spokojenosti se službou. Navržené procesy se opírají zejména o automatizaci, která je jak se ukázalo pro vývoj multiplatformních aplikací vhodná. Testy API byly i přes dané problémy navrženy a implementovány pro všechny implementované funkcionality. Tyto testy odhalily chyby v API, kvůli kterým v době testování nebylo schopné provozu v produkčním prostředí. Implementace testů navíc poukázala na nedostatečnou dokumentaci API, která proces značně stěžovala.

Dle mého názoru byly cíle práce splněny. I když jsou analýza a navržené procesy vztaženy ke službě ElateMe, mají potenciál být šablonou pro obdobné postupy v jiných multiplatformních aplikacích.

Ačkoliv postupy pro zabezpeční QA jsou již navrženy, stále zůstává mnoho nenavržených a neimplementovaných automatizovaných testů pro ostatní komponenty. Zde je prostor pro pokračování v této práci.

Literatura

[1] *Platform.* [cit. 2018-05-10]. Dostupné z: https://www.techopedia.com/definition/3411/platform

[2] *Computer Platforms: Definition, Types & Examples.* [cit. 2018-05-10]. Dostupné z: https://study.com/academy/lesson/computer-platforms-definition-types-examples.html

[3] *Cross platform.* [cit. 2018-05-10]. Dostupné z: https://www.techopedia.com/definition/17056/cross-platform

[4] *Multiplatform.* [cit. 2018-05-10]. Dostupné z: https://techterms.com/definition/multiplatform

[5] *Multiplatformní vývoj aplikací – druhy a nástroje.* [cit. 2018-05-10]. Dostupné z: http://www.krosapp.cz/Blog/multiplatformni-vyvoj-mobilnich-aplikaci-druhy-nastroje

[6] *Launch.* [cit. 2018-05-10]. Dostupné z: https://developer.android.com/distribute/best-practices/launch/

[7] *App review.* [cit. 2018-05-10]. Dostupné z: https://developer.apple.com/support/app-review/

[8] COHEN, M. C. P., David; LINDVALL: Agile software development. *DACS SOAR Report*, 2003.

[9] Continuous Integration. *martinfowler.com*, květen 2006, [cit. 2018-05-03]. Dostupné z: https://www.martinfowler.com/articles/continuousIntegration.html

[10] *Základy systému Git.* [cit. 2018-05-10]. Dostupné z: https://git-scm.com/book/cs/v2/Úvod-Základy-systému-Git

[11] *Making CI Effective in Any Size Organization*. [cit. 2018-05-10]. Dostupné z: http://www.anarsolutions.com/making-ci-effective-size-organization/

[12] *An Introduction to Continuous Integration, Delivery, and Deployment*. [cit. 2018-05-10]. Dostupné z: https://www.digitalocean.com/community/tutorials/an-introduction-to-continuous-integration-delivery-and-deployment

[13] *Continuous integration vs. continuous delivery vs. continuous deployment*. [cit. 2018-05-10]. Dostupné z: https://www.atlassian.com/continuous-delivery/ci-vs-ci-vs-cd

[14] *GitLab Documentation*. [cit. 2018-05-10]. Dostupné z: https://docs.gitlab.com/

[15] *Docker Documentation*. [cit. 2018-05-10]. Dostupné z: https://docs.docker.com/

[16] *Docker Tutorial*. [cit. 2018-05-10]. Dostupné z: https://medium.com/@deshanigeethika/docker-tutorial-a6aa5b41e3ff

[17] *Docker Hub*. [cit. 2018-05-10]. Dostupné z: https://hub.docker.com/

[18] *Graph API*. [cit. 2018-05-10]. Dostupné z: https://developers.facebook.com/docs/graph-api

[19] *Apache JMeter*. [cit. 2018-05-10]. Dostupné z: https://jmeter.apache.org/

Příloha **A**

Seznam použitých zkratek

QA Quality assurance

API Application programing interface

IT Informační technologie

OS Operační systém

UI User interface

HTML Hypertext markup language

JSON Javascript object notation

XSS Cross-site scripting

CI Continuous integration

REST Representational State Transfer

JVM Java virtual machine

XML Extensible markup language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `tests.jmx` ..... zdrojové kódy testů
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF