Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Metodika Continuous Integration při vývoji webové aplikace

Bc. Pavel Fól

Diplomová práce
2018

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Pavel Fól**
Osobní číslo:	**I16167**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Metodika Continuous Integration při vývoji webové aplikace**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem práce je popsat nástroje a postupy používané v rámci metodiky Continuous Integration a využít je při vývoji webové aplikace. V teoretické části práce se autor zaměří na popis problematiky vývoje softwaru v teamovém prostředí s představením nejběžněji používaných nástrojů, které pokrývají celý proces vývoje. Dále autor představí a porovná běžně dostupné nástroje pro CI. V praktické části autor vytvoří webovou aplikaci, při jejímž vývoji využije zvolený nástroj pro CI. Pro vybraný nástroj stručně popíše jeho instalaci a nastavení. Webová aplikace bude analyzovat a vhodným způsobem prezentovat informace, extrahované z databáze. Databáze bude obsahovat komentáře z internetových diskuzí a její plnění není předmětem této diplomové práce. Komentáře budou pocházet ze sběru dat z českých zpravodajských portálů.

Rozsah grafických prací: **10**

Rozsah pracovní zprávy: **60**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

DUVALL, Paul M., Steve MATYAS a Andrew GLOVER. Continuous integration: improving software quality and reducing risk. Upper Saddle River: Addison-Wesley, c2007. ISBN 978-0-321-33638-5.
MARTIN, Robert C. Čistý kód: [návrhové vzory, refaktorování, testování a další techniky agilního programování]. Přeložil Jiří BERKA. Brno: Computer Press, 2009. ISBN 978-80-251-2285-3.

Vedoucí diplomové práce: **Ing. Pavel Jetenský, Ph.D.**
Katedra informačních technologií

Datum zadání diplomové práce: **30. října 2017**
Termín odevzdání diplomové práce: **18. května 2018**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2017

PROHLÁŠENÍ

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 15. 5. 2018 Bc. Pavel Fól

Poděkování

Děkuji Ing. Pavlu Jetenskému, Ph.D. za pomoc a cenné rady při zpracování této práce. Velký dík patří hlavně mé rodině za podporu a trpělivost během studia. Děkuji také všem přátelům a Ing. Kateřině Tauchmanové za rady při zpracování práce.

ANOTACE

Diplomová práce se zabývá popisem kontinuální integrace a jejím exemplárním nasazením při vývoji webové aplikace. V rámci teoretické části je popsán proces vývoje softwaru, s důrazem na představení kritických částí pro správné pochopení popisované metodiky, jako jsou verzovací systém, buildy a testování aplikace. Praktická část zahrnuje vývoj webové aplikace pro porovnávání podobnosti mezi texty. Důraz je kladen na popis toho, jak byla kontinuální integrace v rámci vývoje využita spolu s popisem reprodukce řešení pomocí serveru Jenkins CI.

KLÍČOVÁ SLOVA

kontinuální integrace, Jenkins, Scrum, Spring Boot, Kotlin

TITLE

Continuous integration as an approach during development of web application

ANNOTATION

The aim of this thesis is to describe continuous integration and its exemplary deployment in connection with development of web applications. In the theoretical part of the thesis, software development process is described, with an emphasis on introducing the critical parts for correct understanding of the described methodics, such as versioning system, builds and application testing. The practical part of the thesis includes developing a web application for comparing similarities between texts. Emphasis is placed on describing how continuous integration was used in terms of development, along with a description of reproducing the solution with a Jenkins CI server.

KEYWORDS

Continuous Integration, Jenkins, Scrum, Spring Boot, Kotlin

OBSAH

SEZNAM TABULEK, GRAFŮ, ILUSTRACÍ A ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK A ZNAČEK

CI	Continuous Integration
SCM	Software Configuration Management
VCS	Version System Control
IDE	Integrated Development Environment
SDK	Software Development Kit
XP	Extreme Programming
CR	Code Review
UAT	User Acceptance Testing
PaaS	Platform as a Service
AWS	Amazon Web Services
JVM	Java Virtual Machine
JDK	Java Development Kit
REST	Representational State Transfer
API	Application Programming Interface
GUI	Graphical User Interface
NPE	NullPointerException

TERMINOLOGIE

Code review (CR) je událost, při které programátor kontroluje kód toho druhého s cílem odhalit případné chyby. Několikrát bylo také dokázáno, že urychluje a zefektivňuje proces vývoje softwaru. (SMARTBEAR SOFTWARE, 2018a)

Platform as a service (PaaS) je kategorie cloud computingu, která poskytuje platformu a prostředí které dovoluje vývojářům a firmám vyvíjet aplikace a používat služby, které jsou hostovány v rámci cloudu. Zpravidla se platí za zprostředkované zdroje. (INTEROUTE, 2018)

Software deploy je proces složen z několika kroků, jejichž cílem je nasazení aplikace do produkčního prostředí.

Java Virtual Machine je abstraktní virtuální počítač, díky konkrétním instancím je možné na počítači spouštět Java aplikace. (VENNERS, 2018)

Úvod

Vývoj softwaru se dotýká nějakým způsobem každého z nás. Software je totiž naprosto všude. Především díky vývoji této disciplíny jsou výsledné produkty stále více komplexnější a jejich vývoj je sofistikovanější. Ať už v domácích spotřebičích, v každodenně používaných chytrých telefonech nebo jako produkt, díky kterému komunikuje klient se svou bankou, všude je nějaký software nasazen. Aby byl výsledný produkt širokou veřejností přijat, musí být mimo jiné i kvalitní. Mezi ukazatele kvality je možné zařadit například i dodávání nových verzí s novými vlastnostmi a opravami nalezených chyb.

V první části práce jsou popsány agilní metodiky, díky kterým v dnešní době většina softwaru vzniká iterativně. Během vývoje je tak nutné zajistit, aby byl software pokryt různými druhy testů, díky kterým získává vývojový tým zpětnou vazbu, zdali nová funkcionalita neovlivnila již vyvinutou část. Tyto testy a jejich důležitost pro následné téma kontinuální integrace jsou součástí teoretické části.

Kontinuální integrace (anglicky Continuous Integration) je v dnešní době velice důležitý přístup při vývoji softwaru, který při správném nasazení podporuje práci vývojářů v týmu a při každé změně v kódu vygeneruje zpětnou vazbu o aktuálním stavu. Je také prvotním předpokladem pro adaptaci stále více se rozšiřujících se praktik Continous Delivery a Continous Deployment. Pro podporu kontinuální integrace existuje mnoho nástrojů, přičemž některé z nich jsou popsány na závěr teoretické části, stejně tak jako samotné téma kontinuální integrace.

V rámci praktické části je vyvíjena webové aplikace pro vyhledávání podobnosti mezi komentáři získaných z některých českých zpravodajských portálů. Pro vývoj aplikace je použit framework Spring Boot a programovací jazyk Kotlin. Aplikace je během vývoje kontinuálně integrována pomocí integračního serveru Jenkins. Důraz v praktické části je kladen jednak na popis samotné aplikace a použité technologie, tak i na popis konkrétního kontinuálního procesu a reprodukci jeho nasazení. Závěr praktické části je věnován kapitole prezentující statistiky o podobnosti mezi komentáři, spolu s demonstrací vybraných podobných komentářů.

1 LITERÁRNÍ REŠERŠE

V dnešním světě, do kterého jsou plně integrovány digitální technologie, je v nejednom odvětví velká konkurence. Jsou to právě digitální technologie, které jsou hnány vysokým tempem dopředu a náročnost jejich uživatelů vzrůstá. Díky internetu tak může každý začít s vývojem své aplikace, tu může volně vystavit pomocí distribučních kanálů a začít tak profitovat. V případě jednotlivce jde jen o to, jak je člověk zdatný se učit programovat a jak je jeho nápad na obsah aplikace unikátní. Ve většině případů je však software vyvíjen ve větších týmech. Ať už je software vyvíjen jednotlivcem nebo v týmu, většinou prochází stejnými etapami. Pokud se jedná o software, který je neustále vyvíjen a je dodáván kontinuálně, začnou se tyto etapy opakovat.

O evoluci softwaru se zmiňuje ve své knize například Grady Booch, kde již v roce 1991 zmiňuje právě toto spojení „continuous integration“ (dále jako CI). To je mimo jiné také první použití tohoto slova ve významu, v jakém bude chápán v celé této práci. V rámci interního prostředí firmy by mělo vzniknout během vývoje několik interních vydání softwaru. Právě vydávání těchto verzí označuje jako princip využívání CI. Stejný princip by se také měl aplikovat na testování. (BOOCH, 1994)

V rámci dalších let se vývoj softwaru posouvá vpřed. Jsou známy nové „best practises“[1], existují nové nástroje pro ulehčení práce a nové počítače již zvládají náročnější úlohy za mnohem kratší dobu. Většina z těchto praktik používaných při vývoji softwaru již zahrnuje i metodiky, které dnes známe pod označením CI. O abstraktní popis CI se v roce 2000 pokusil Martin Fowler. Fowler se ve svém článku snaží popsat, co to vlastně CI je, a jaké může mít benefity nejen pro tým, ale i pro celou společnost. Zjišťuje, že se jedná vlastně o jednu z praktik, která se využívá při extrémním programování, jejímž cílem je integrovat aplikaci několikrát denně. Počínaje vývojářem, který nahraje své změny do centrálního úložiště, přes automatický build a spuštění testů, po úspěšný deployment do produkčního prostředí. Toto je však hrubá kostra CI, která je podpořena dílčími kroky, které je nutné provést. (FOWLER, 2000)

V roce 2006 vydává Martin Fowler revizi svého prvotního textu a dále formuluje myšlenky o CI na základě získaných zkušeností z předchozích let. Adaptace těchto myšlenek a jejich provádění je možné i bez specializovaných nástrojů, avšak je více než užitečné nějaký z nich využít. Fowler tyto nástroje označuje jako servery pro CI. První takový byl Cruise, později přejmenován na CruiseControl. Tento nástroj byl vyvinut firmou ThoughtWorks, ve které sám

[1] Osvědčené postupy.

Fowler pracuje a sám se i podílel na jeho vývoji. V rámci použití CI serveru již začíná být nutné vymezit pravidla, jak vlastně CI nasadit a používat. Je nutné mít jedno centrální místo, které je zdrojem i cílem zdrojových kódů. K tomu se využívá „Version Control Software“, dále jako VCS. Dále je nutné automatizovat build aplikace a také definovat, co je součástí buildu. Nad buildy je nutné spouštět automatické testy. Každý vývojář by měl nahrávat své lokální změny v kódu do VCS co nejčastěji, při nejmenší četnosti však alespoň jednou denně. Díky tomu je tak možné efektivněji předejít konfliktům[2] ve zdrojovém kódu, případně je tím i zkrácena doba, za jakou je konflikt vyřešen, pokud již nastane. Každý nový příspěvek do VCS by měl také dát vzniknout novému životnímu cyklu buildu na CI serveru, který by na svém konci měl být označen jako úspěšný. Pokud se tomu tak nestane, je nutné ihned zjistit příčiny a napravit je. Aby byl celý proces efektivní, musí být build dost rychlý. Pro ještě lepší zpětnou vazbu by měl být software testován v prostředí, které se co nejvíce blíží produkčnímu. Výsledky a informace o kvalitě softwaru by měly být dostupné všem, kteří jsou zainteresovaní na jeho vývoji. (FOWLER, 2006)

Ve stejném roce přispívá kolektiv autorů článkem nazvaným „The Deployment Production Line“ na konferenci Agile Conference. Na základě získaných zkušeností prezentují, jak k CI přistupovat. Dalo by se říci, že v tomto roce již existuje stabilní definice toho, co CI je, jaké má výhody a jak metodiky uchopit a nasadit ve svém vlastním prostředí. (HUMBLE, NORTH a READ, 2006)

Díky tomu vzniká kniha s výstižným názvem „Continuous Integration – Improving Software Quality and Reducing Risk“, ve které autoři kromě samotné definice také jednotlivé kroky prakticky demonstrují. Začíná se také mluvit o nástrojích typu Ant nebo Maven, díky kterým je celý proces ještě o něco jednodušší. Důraz je kladen také na to, jak se vypořádat se zdroji dat aplikace. (DUVALL, MATYAS a GLOVER, 2007)

V dnešní době existuje již množství nástrojů, které jsou přímo určeny pro nasazení a podporu CI. V této práci, a hlavně její praktické části, je využit nástroj Jenkins, který patří ke známějším a je velmi dobře popsán v rámci uživatelské příručky. (JENKINS, 2018)

Během jednotlivých fází v rámci CI prochází aplikace také testováním. Testování se dělí do několika kategorií, přičemž je více než chtěné automatizovat co nejvyšší počet testů. A to je právě velice důležité pro nasazení CI. Testy, které se spouštějí při každém buildu, jsou schopné indikovat funkčnost softwaru v každý okamžik změny. Díky tomu lze nastavit různá pravidla

[2] Konflikt může nastat, pokud dva nebo více vývojářů provede změny na stejném řádku ve stejném souboru.

pro notifikace. Uživatel má tak téměř ihned k dispozici informace, zdali jeho změny například nezpůsobily nefunkčnost části nebo celého softwaru. Popisem testování se zabývá Ron Patton, který v roce 2002 vydává aktualizované vydání knihy Software Testing. (PATTON, 2001)

V rámci akademických prací se již skupina autorů zabývá procesem kontinuální integrace. Například Matúš Zamborský (2012) se ve své práci věnuje identifikaci a optimalizaci procesů s nasazením kontinuální integrace do vývojového procesu. Martin Hujer (2012) se pak snaží popsat nasazení kontinuální integrace v rámci aplikace v PHP.

Tato práce si klade za cíl informovat čtenáře zprvu o tom, s čím se vlastně v rámci vývoje software setká. Ať už to jsou nástroje využívané při vývoji anebo i metodiky, kterými se při vývoji tým lidí řídí. Důležitým úvodem před popisem kontinuální integrace je nutná zmínka o tom, jak funguje vývoj v teamu několika lidí, kteří pracují nad stejným kódem. Popsané postupy a výhody kontinuální integrace jsou exemplárně předvedeny nad reálnou webovou aplikací postavenou nad sadou nástrojů Spring Boot s využitím jazyka Kotlin. Aplikace si klade za cíl analyzovat podobnosti textů a vhodnou reprezentaci výsledků v rámci responzivní webové aplikace. Tato problematika není přímo obsahem práce, každopádně se jedná o velice zajímavou disciplínu. Mezi důležité publikace této problematiky patří kniha Algorithms on Strings, Trees, and Sequences od Dana Gusfielda. (GUSFIELD, 1997)

Autor této práce identifikuje dva hlavní přínosy. Technologie, které jsou využity v rámci praktické části, jsou moderní a čím dál častěji vyhledávané. Jedná se hlavně o Spring Boot a programovací jazyk Kotlin. Praktická ukázka toho, jak tyto technologie úspěšně nasadit v rámci CI serveru Jenkins, je jednoznačně přínosem, jelikož se jedná o vcelku nepopsanou oblast, která je tak nyní pokryta. Díky vzniknuvší webové aplikaci, jež autor pojmenoval Zoumi.cz, si může čtenář ověřit, jestli v některé z diskuzí, ke které má aplikace přístup, neexistují takové komentáře, jež jsou označeny jako podobné. V době, kdy se stále častěji mluví o snaze ovlivnit veřejné mínění, se může jednat například o komentáře, jež jsou psány tou samou osobou (nebo skupinou osob), avšak pod jiným jménem s částečně pozměněným kontextem. Aplikace tak může sloužit čtenáři jako jakýsi indikátor, kdy by měl zbystřit a dávat větší pozor.

2 VÝVOJ SOFTWARU

Na začátku všeho nového je vždy myšlenka. Nejinak je tomu i v doméně vývoje softwaru. Pokud vzniká nový software, vzniká na základě něčí myšlenky. Díky programovacím jazykům, podpůrným frameworkům a nástrojům pro vývoj není v dnešní době problém vytvořit téměř jakýkoliv program. Při vývoji je vhodné se řídit některou z popsaných metodik, které definují nejen pravidla a postupy, ale také například udávají, jak se stavět k plánování a řízení průběhu vývoje. To má za následek hlavně efektivitu při vytváření softwaru. Nelze jednoduše říci, jaká metodika je nejlepší pro daný projekt a měla by se následovat. Ovšem metodika je jakási šablona, ze které je možné si převzít pouze některé části, není tak nutné se řídit pravidly přesně.

2.1 METODIKY VÝVOJE SOFTWARU

V následujících kapitolách se čtenář seznámí s některými z tradičních metodik a následně také se zástupci agilních metodik. Základní přehled by čtenáři měl pomoci pochopit to, proč je CI tak důležitá při následování agilních metodik.

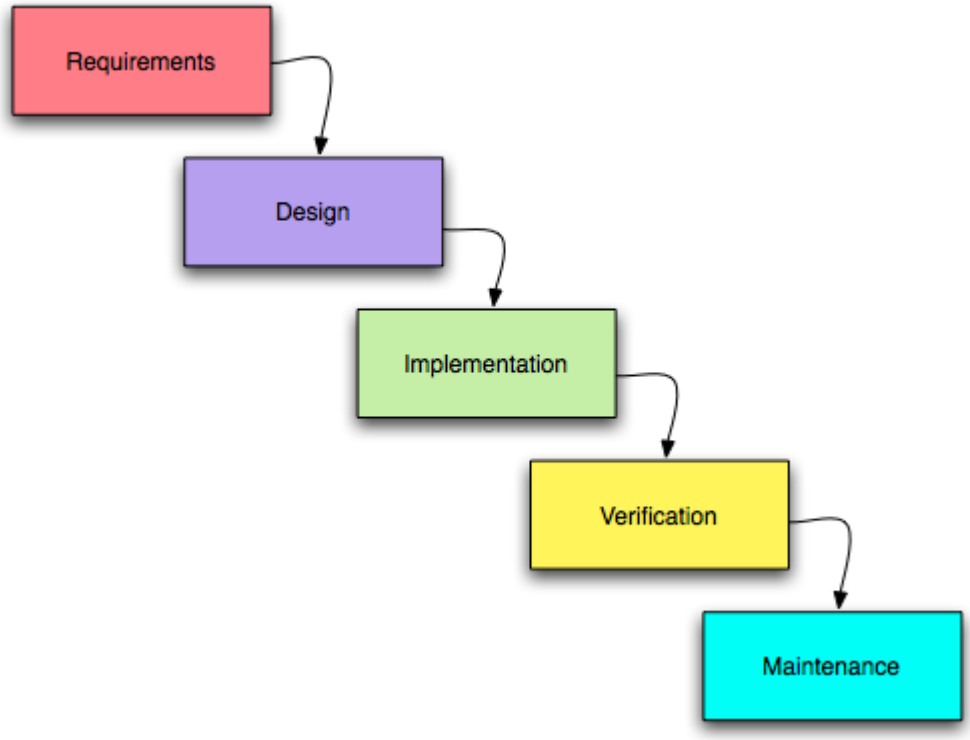
2.1.1 KLASICKÉ METODIKY

Mezi klasické metodiky patří například vodopádový model, spirálový model nebo Rational Unified Process. Pro přiblížení bude popsán první zmíněný a jeho modifikace. (KADLEC, 2004)

2.1.1.1 Waterfall

Jedná se o model, jehož základem je lineární sekvenční přístup k řešení jednotlivých částí životního cyklu. V rané fázi se definují požadavky, po kterých se přejde k návrhu, dále na samotnou implementaci, kdy je software později ověřen a následně udržován. Mezi těmito fázemi je tedy definovaný tok, kdy lze přejít na následující fázi, pouze pokud je kompletně dokončena fáze předchozí (díky tomu se tedy označuje jako vodopád). Tento model byl v roce 1970 formálně popsán v článku „Managing the Development of Large Software Systems“, jejž vydal Winston W. Royce. V tomto článku ještě není označen jako „waterfall“, nicméně již autor tento přímý přístup označuje jako riskantní. Typický spádový model je na obrázku 1. (ROYCE, 1970)

Takový přístup se dal v dřívějších dobách využívat, protože většina vytvářených systémů byla vymyšlena přímo programátory s pouze malými nebo žádnými náměty od stakeholderů. Samotní uživatelé se do vývoje nezapojují, takže ani požadavky na změny po specifikaci téměř nepřicházejí. (LAPLANTE a NEILL, 2004)



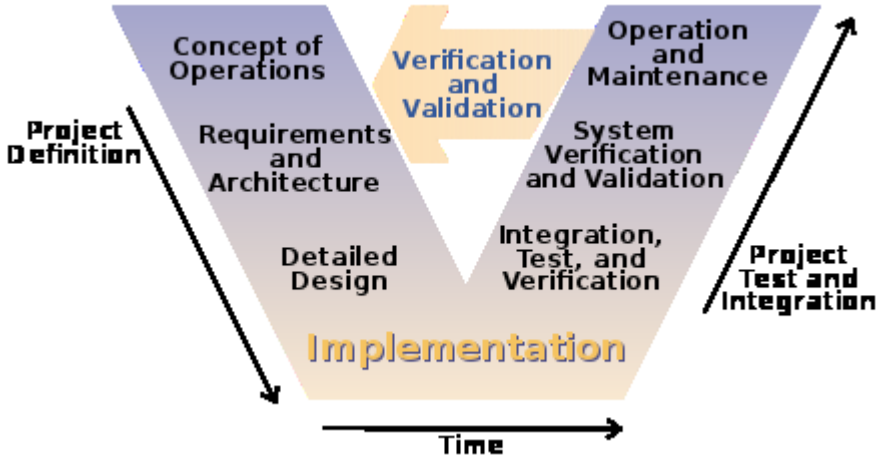
Obrázek 1: *Schéma vodopádového modelu*[3]

2.1.1.2 V-Model

Takzvaný „V-Model“ je rozšířením předchozího vodopádového modelu. Písmeno „V“ v názvu tohoto modelu odráží přímo strukturu, jak je model postaven. Struktura modelu je vidět na obrázku 2. Na rozdíl od předchozího modelu však po fázi implementace nepokračuje lineárně směrem dolů, nýbrž se „odráží“ a pokračuje směrem vzhůru. Důvodem, proč se tomu tak děje, je to, že každá fáze je testovatelná. Implementační fázi tak odráží jednotkové testování (verifikace), správnost návrhu je otestována pomocí integračních testů a takto se pokračuje

[3] Zdroj: (WIKIMEDIA COMMONS, 2005a).

směrem vzhůru. Přístup k testování je nejdůležitější rozdíl oproti vodopádovému modelu. (ISTQB EXAM CERTIFICATION, 2013)



Obrázek 2: *Schéma V-modelu*[4]

2.1.2 AGILNÍ METODIKY VÝVOJE SOFTWARU

Předchozí zmíněné metodiky mají sice své zastánce, nicméně jsou těžkopádné a například reakce na změnu v zadání není nemožná, ale postihne celý proces. Pokud se bude vývojový tým řídit metodikou vodopádu a zákazník se dostaví ve fázi verifikace s tím, že do programu potřebuje přidat nějakou funkcionalitu, nastává situace, kdy ani jedna strana nebude spokojená. Vývojový tým musí začít od první fáze a integrace, byť jen malé změny, může změnit podstatnou část návrhu. Další čas a úsilí se tak promítne do financí zákazníka. Oproti tomu tak stojí agilní metodiky, které mění přístup k organizaci práce. V únoru roku 2001 vznikl dokument zvaný „The Agile Manifesto“, který uvádí 4 klíčové hodnoty a 12 principů, jak přistoupit k řízení vývoje softwaru. Klíčové hodnoty jsou následující: (AGILE MANIFESTO, 2001)

1. jednotlivci a interakce před procesy a nástroji,
2. fungující software před vyčerpávající dokumentací,
3. spolupráce se zákazníkem před vyjednáváním o smlouvě,
4. reagování na změny před dodržováním plánu.

Mezi autory patří například Kent Beck, jeden z autorů agilního procesu zvaného extrémní programování. Dále také Martin Fowler, o němž již byla zmínka dříve a patří mezi hlavní osobnosti zabývající se tématem kontinuální integrace.

[4] Zdroj: (WIKIPEDIA COMMONS, 2005b).

Agilní metodiky jsou založeny na několika principech. Software je například dodáván v krátkých intervalech a díky tomu tak může zákazník do procesu zasáhnout. Krátký interval není přesně definován, ale mělo by se jednat o týdny, maximálně měsíce. Tyto intervaly se pak obecně nazývají iterace. Všechny změny od zákazníka jsou vítány a vzhledem k použité metodice jsou i bez problému proveditelné. Preferuje se osobní komunikace, a to jak v rámci týmu, tak i ze strany zákazníka. Ten se proaktivně angažuje v projektu a spolupracuje po celou dobu s vývojovým týmem. Důležitým prvkem je také samoorganizující tým, který je schopný se organizovat znovu a znovu a přijímat různé výzvy, kterým musí čelit. To je jen krátký výčet principů obsažených v manifestu. (AGILE MANIFESTO, 2001) Highsmith a Cockburn o agilních metodikách napsali: (HIGHSMITH a COCKBURN, 2001)

„What is new about agile methods is not the practices they use, but their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness and maneuverability. This yields a new combination of values and principles that define an „agile" world view."[5]

2.1.2.1 Extrémní programování

Jako první ze zástupců agilních metodik je uvedeno extrémní programování, dále jako XP. Autorem tohoto přístupu je Kent Beck, který tuto metodiku vyvíjel a pracoval na ní během devadesátých let, kdy pracoval v Chrysleru na výplatním systému. Jelikož jsou zde všechny používané praktiky vyhnané do extrému, je v názvu použito slovo „extrémní". Hlavní praktiky v extrémním programování jsou:

- plánovací hra,
- krátká doba mezi verzemi,
- metafora,
- jednoduchý návrh,
- testy,
- refaktorování,
- párové programování,
- kontinuální integrace,
- společné vlastnictví,
- zákazník v místě,

[5] Vlastní překlad: Na agilních metodikách není nové to, jaké praktiky používají, ale jak vnímají lidi jako hnací sílu k úspěchu projektu spolu s intenzivním zaměřením na efektivitu a manévrovatelnost. To přináší novou kombinaci hodnot a principů, které definují pohled na agilní svět.

- 40 hodinový týden,
- otevřený prostor,
- prostě pravidla.

Plánovací hra

Zákazníci určují rozsah a čas release cyklu[6] na základě výpočtů od programátorů. Programátoři pak implementují pouze ty funkcionality, které byly zaplánovány do aktuální iterace.

Krátká doba mezi verzemi

Je nutné vydávat nové verze softwaru v tak krátké době, jak je to jen možné (například denně). Díky tomu je zajištěna rychlá zpětná vazba od zákazníka.

Metafora

Tvar systému je definován pomocí metafor, které jsou sdíleny mezi programátory a zákazníky. Díky tomu je zajištěna vysoká míra pochopení problematiky.

Jednoduchý návrh

Počítá se, že požadavky na systém se budou měnit, proto se vždy vytváří pouze a jen to, co se požadovalo. Nikdy se nevytváří nic navíc jenom proto, že by to někdy mohlo být potřebné.

Testy

Programátoři píší co nejvíce jednotkových testů. Tyto testy jsou psány ještě dříve, než se začne programovat. Zákazníci tvoří své akceptační testy. Všechny testy musí při každém spuštění procházet.

Refaktorování

Návrh systému prochází vývojem a je tak jisté, že bude nutné změnit i kód. Pokud programátor narazí na kód, který lze optimalizovat při zachování funkčnosti, měl by tak učinit.

Párové programování

Kód je psaný dvěma osobami, přičemž obě sdílejí jeden počítač. Prakticky tedy jeden programuje a druhý přemýšlí. Programátoři mezi sebou vedou diskuzi a v dohodnutých intervalech se mohou vystřídat.

[6] Release cyklus je suma všech součástí vývoje softwaru počínaje od začátku vývoje do vydání nové verze.

Kontinuální integrace

Je nutné, aby byl každý nový kód integrován s aktuálním systémem v co nejkratší době (v rámci hodin). V rámci integrace musí úspěšně projít všechny testy. Pokud tomu tak není, změny musí být zrušeny a opraveny.

Společné vlastnictví

Každý z programátorů odpovídá za celý kód. Pokud je možnost kód kdekoliv vylepšit, učiní tak.

Zákazník v místě

Zákazník je v průběhu vývoje součástí týmu a je přítomen v místě, kde tým pracuje.

40 hodinový pracovní týden

Pro XP je důležité, aby měl pracovní týden přesně 40 hodin. Časté přesčasy indikují možnost problému, který musí být vyřešen.

Otevřený prostor

Tým pracuje ve větší místnosti a ta dvojice, která párově programuje, si sedá nejlépe doprostřed místnosti.

Prostě pravidla

Tým pracuje podle pravidel, které si nastaví sám. Tato pravidla se mohou kdykoliv přizpůsobit a měnit. (BECK, 1999)

2.1.2.2 Scrum

Scrum je další z řady agilních metodik pro produktivní vývoj softwaru a doručování produktu v nejvyšší možné kvalitě. Jedná se pravděpodobně o jednu z nejvíce používaných metodik, u jejíhož zrodu stáli pánové Ken Schwaber a Jeff Sutherland a jejíž historie se datuje do 90 let. Scrum team se skládá z menšího počtu lidí, kteří mají různé role. Vývoj probíhá iterativně v krátkých intervalech, do kterých jsou předem naplánované úlohy. Žádné další úlohy do těchto intervalů nepřibývají, pokud se tak nerozhodne tým jako celek. Existuje několik ceremonií, přičemž některé jsou každodenní a některé jsou buď na začátku nebo na konci iterace. Iterace se nazývají sprinty a jsou klíčové pro Scrum. Jak již bylo zmíněno, team se skládá z několika rolí, a to z vlastníka produktu, vývojového týmu a Scrum mastera. Oba výše zmínění autoři sepsali příručku „The Scrum Guide“, která přesně definuje Scrum. (SCHWABER, SUTHERLAND, 2017)

Vlastník produktu je osoba, která je na straně obchodu a prezentuje zájmy zákazníka. Jako jediný se stará o správu backlogu produktu. Backlog je ve své podstatě seznam všeho, co je požadováno, aby produkt uměl. Tento seznam je dynamický a často se mění, do backlogu patří zejména:

- všechny vlastnosti,
- funkce,
- požadavky,
- vylepšení,
- opravy, které jsou naplánovány do budoucích verzí.

Vývojový tým je skupina osob, která se stará o dodání požadovaných úloh. Jedná se o samoorganizující tým, jež pracuje v rámci iterací nazvaných sprint. Do sprintu se vždy naplánují úlohy z produktového backlogu a to tak, aby bylo zajištěno jejich včasné dodání. Velikost týmu není přesně dána, ale mělo by se jednat o skupinu 4-8 lidí.

Scrum master je člověk, který stojí mezi vlastníkem produktu, vývojovým týmem a organizací. Ve vztahu vůči vlastníkovi produktu se zabývá tím, zda veškeré vytyčené cíle jsou správně chápány napříč celým týmem. Zároveň pomáhá celému týmu chápat produktový backlog a také dohlíží nad správnou adaptací agilních metodik. Vývojovému týmu pomáhá například dodržovat samoorganizovanost nebo odstraňuje překážky při vývoji. V rámci organizace pomáhá s plánováním a nasazením Scrumu.

Během každého sprintu se pracuje na těch úlohách, které byly vybrány z produktového backlogu a naplánovány, případně na úlohách, které se nestihly ve sprintu předchozím. Tyto úlohy jsou v rámci sprintu uchovány ve sprint backlogu. V rámci sprintu se tak stav jednotlivých úkolů mění a je nutné si v rámci týmu definovat, kdy lze označit úlohu jako dokončenou (je ve stavu „done“). Po konci každého sprintu by tak mělo být možné sestavit novou verzi systému, která bude obsahovat všechny nové změny a verze bude s těmito změnami funkční. Tým by měl používat označení „done“ ve chvíli, kdy je zajištěna odpovídající kvalita i po změně v kódu softwaru, tzn. jedná se o naimplementovanou a plně otestovanou úlohu. Pokud na jednom systému pracuje více týmu aplikujících Scrum, pak je pro zajištění vysoce kvalitního systému nutné pracovat se stejnou definicí pro „done“ napříč těmito týmy.

SPRINT PLANNING

Na začátku každého nového sprintu se členové týmu sejdou a společně plánují práci pro toto období. Maximální doba pro plánování by neměla přesáhnout 8 hodin pro sprint trvající měsíc,

pro kratší iterace se pak poměrově zmenšuje. Během plánování je nutné odpovědět na dvě klíčové otázky:

- Co je možné v rámci Sprintu stihnout?
- Jak vybrané úlohy vyřešit?

Co je možné v rámci Sprintu stihnout?

V rámci této části tým pracuje s produktovým backlogem, ze kterého vybírá úlohy. Vybírání probíhá na základě znalosti vývojové kapacity, kterou tým disponuje, a na základě odhadu náročnosti úloh. Je pouze na týmu, kolik úloh si do sprintu naplánuje.

Jak vybrané úlohy vyřešit?

Ve druhé části plánování si tým musí ujasnit, jak bude vybrané úlohy řešit. To zahrnuje například tvorbu designu systému. Pokud tým nedisponuje dostatečnými znalostmi z technické domény řešené úlohy, může si pozvat a poradit se s dalšími odborníky.

DAILY SCRUM

Každý den během sprintu se tým sejde na krátkou schůzi, která se nejčastěji označuje jako stand-up. Pravidla pro tuto schůzi jsou jednoduchá, měla by se konat vždy ve stejný čas na stejném místě. Během této schůze každý člen ostatním nejčastěji sdělí, na čem pracoval předchozí den, na čem bude pracovat nebo pracuje dnes, a případně jestli existuje něco, co mu brání v práci a potřebuje pomoci. Díky tomu je tak každý v týmu informován o aktuálním dění při vývoji úloh, které jsou naplánované ve sprintu.

SPRINT REVIEW

Jedná se o informační schůzku, na které je představeno, co se v rámci sprintu stihlo dodat a co se nestihlo. Realizační tým také odpovídá na otázky, které jsou kladeny k realizaci jednotlivých úloh. Mezi účastníky této schůzky je celý realizační team, vlastník produktu a klíčové osoby všech zúčastněných stran.

SPRINT RETROSPECTIVE

Jedná se o schůzku, která následuje po „Sprint Review“ a naopak se koná před dalším „Sprint Planning“. Účastní se jí všichni členové týmu a Scrum master. Jedná se o příležitost v rámci týmu zjistit, co se lidem v rámci sprintu líbilo a kde vidí případnou příležitost pro vylepšení. Schůzka by se měla nést v pozitivním duchu a měla by být produktivní. Na konci by měly být identifikovány body na zlepšení, které budou učiněny v rámci příštího sprintu.

2.2 NÁSTROJE POUŽÍVANÉ PRO VÝVOJ

V této části je proveden pomyslný skok z metodik rovnou k tomu podstatnému při vytváření nového softwaru, a to je samotný vývoj. Ze začátku je nutné uvést, že k vývoji nového softwaru bude každý využívat různé nástroje. Jaké nástroje jsou vybrány, záleží na několika aspektech, například podle programovacího jazyka bude zvoleno vývojové prostředí, dále jako IDE. Autor práce z vlastní zkušenosti dodává, že je vhodné při vývoji v týmu využívat stejných nástrojů. Kromě toho, že je možné sdílet různé vlastní nastavení nebo řešení společných problémů, tak je také jednodušší například při code review, případně při párovém programování adaptace toho druhého na vývojové prostředí kolegy. Avšak ty nejzákladnější nástroje, se kterými většina vývojářů (nejen v týmovém prostředí) pracuje, jsou následující:

- vývojové prostředí – IDE,
- verzovací systém,
- textový editor,
- SDK.

2.2.1 VÝVOJOVÉ PROSTŘEDÍ

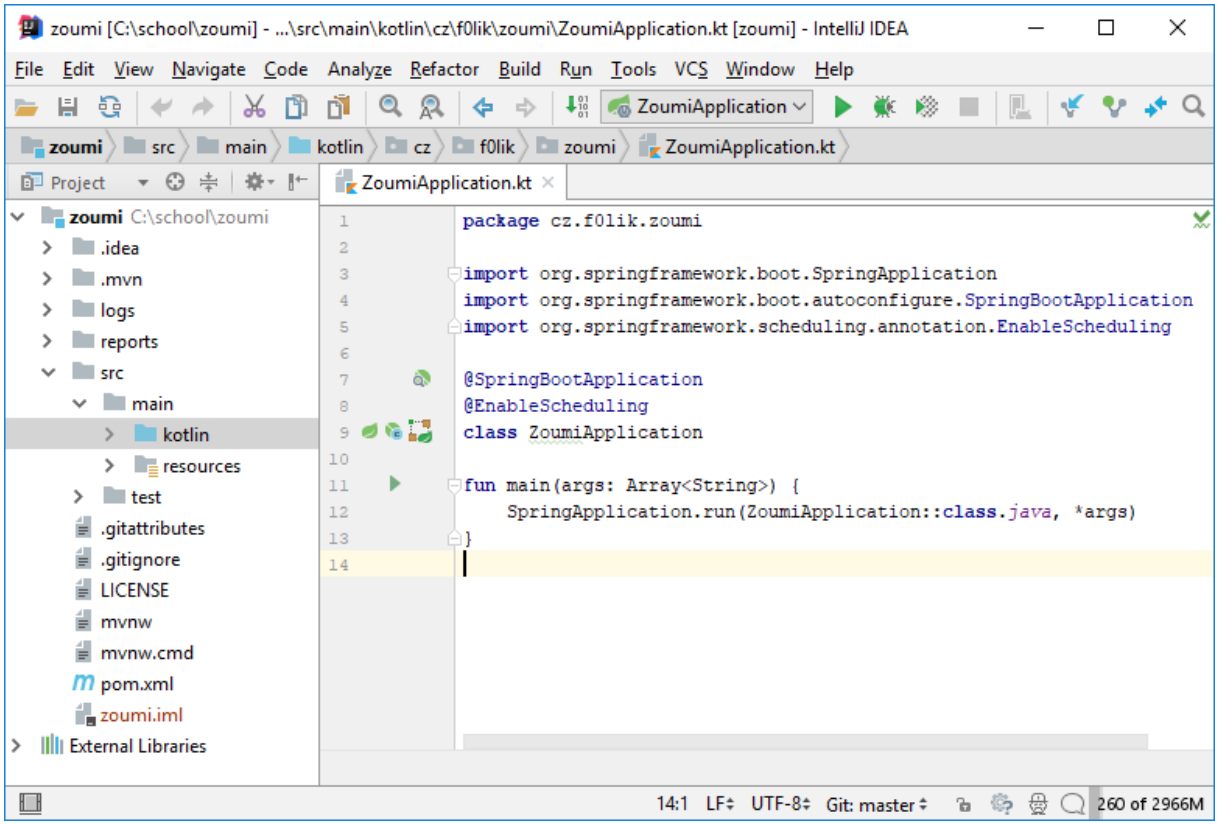
Vývojové prostředí je typicky sada nástrojů, ve kterých vývojář píše a testuje vyvíjený software. Typické označení, které se používá pro tuto sadu, je IDE. Základním prvkem je prohlížeč souborů projektu, textový editor kódu podporující danou syntaxi jazyka, kompilátor nebo interpreter a debugger. Právě zmíněná integrace všech těchto nástrojů odlišuje IDE od samotného textového editoru. Některá prostředí podporují více programovacích jazyků, jiná jsou zaměřena pouze na úzkou doménu. V dnešní době jsou IDE již robustní nástroje, díky kterým je možné efektivně a rychle vyvíjet téměř bez potřeby využívat další nástroje. Pro týmovou práci například podporují integraci s nejpoužívanějšími verzovacími nástroji a obsahují i nástroje pro řešení konfliktů v kódu. Také je možná přímá integrace na některý z trackovacích nástrojů, pro příklad IntelliJ IDEA ve spojení s YouTrack téměř eliminuje nutnost otevírání webového prohlížeče a přechodu na webové rozhraní trackovacího nástroje.

Na základě toho, jak jsou jednotlivá IDE vyhledávána ve vyhledávači Google, se Pierre Carbonnelle snaží sestavit tabulku zvanou „TOP IDE“, ve které mapuje pořadí pro dané IDE podle toho, jakou zaujímá část ve vyhledávání a jaký je trend vyhledávání oproti stejnému měsíci loňského roku. V tabulce 1 jsou uvedena data aktuální pro březen 2018 (a trend oproti březnu 2017). (PIERRE CARBONNELLE, 2018)

Tabulka 1: *Vyhledávanost IDE v březnu 2018*

Pořadí	IDE	Podíl	Trend
1.	Visual Studio	26,03 %	+1,0 %
2.	Eclipse	24,19 %	-2,9 %
3.	Android Studio	10,8 %	+0,3 %
4.	NetBeans	6,92 %	-0,3 %

Autor v rámci praktické části této diplomové práce využívá IntelliJ IDEA, která se dle výše zmíněného zdroje v rámci měsíce března umístila na 8. příčce s podílem ve vyhledávání 4,14 %. Na obrázku 3 je základní rozvržení částí tohoto IDE s otevřeným projektem praktické části této diplomové práce, tedy aplikací Zoumi.cz (více v kapitole 4).



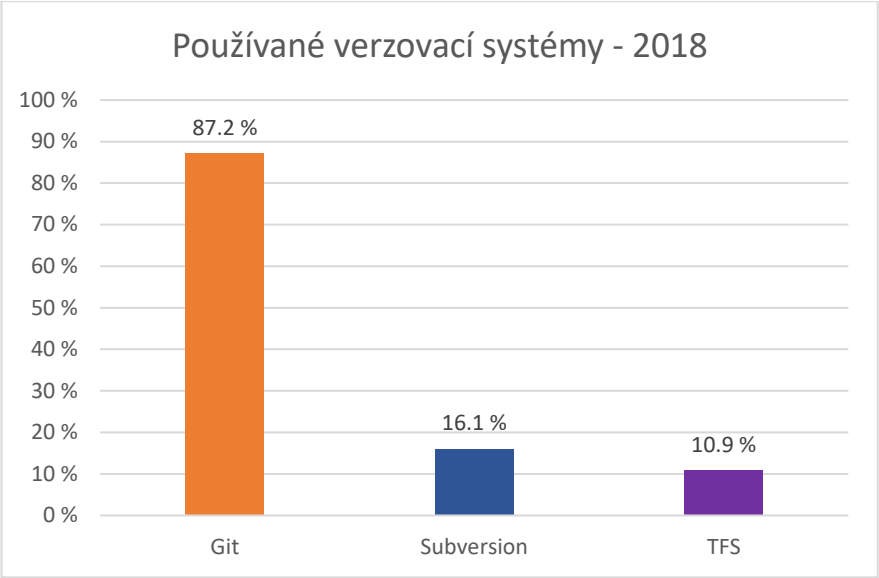
Obrázek 3: *IntelliJ IDEA s otevřeným projektem aplikace Zoumi.cz*[7]

[7] Zdroj: vlastní.

2.2.2 VERZOVACÍ SYSTÉM

Pro vlastní aplikaci metodik CI je nutné pracovat s nejpodstatnější částí, což je právě verzovací systém. Hlavní vlastností verzovacího systému je správa změn ve zdrojovém kódu aplikace. Tento systém je středobodem všech změn ve zdrojovém kódu aplikace, ke kterému přistupují všichni vývojáři. Ti do něj nahrávají své upravené soubory. Díky tomu si tak ostatní lidé mohou tyto změny stáhnout a pracovat nad aktuální kopií aplikace. Verzovací systém také umožňuje vracení se v čase a získávání různých verzí zdrojového kódu a dalších souborů. Výhodou je také to, že verzovací systémy podporují takzvané větve. Z hlavní vývojové linie tak lze vytvořit identickou, ale oddělenou větev. To může přijít vhod v situacích, kdy je třeba upravit podstatnou část aplikace, a mohlo by se stát, že v delším časovém horizontu by byla aplikace nefunkční. V takovém případě se vyvíjí do větve, která je po úspěšném vývoji zase připojena k hlavní programové větvi (a jsou tak přeneseny změny). Mezi další jejich využití pak patří například to, kdy je aplikace vydávána ve verzích. Každá hlavní verze může mít vlastní očíslovanou větev a aktuální verze bude vždy vyvíjena do hlavní větve.

Verzovacích systémů existuje několik, a i když je hlavní myšlenka správy změn v kódu stejná, mohou pracovat na odlišných principech, případně využívají odlišné operace pro stejné úlohy. Mezi nejznámější a nejpoužívanější patří Git a Subversion, přičemž první jmenovaný jednoznačně vede. Portál StackOverflow každý rok vydává různé statistiky, jež jsou výstupem dotazování vývojářů na různé otázky. V roce 2018 již disponuje odpověďmi od více než 100 000 respondentů. Ti mimo jiné odpovídají i na otázku, s jakými verzovacími systémy nejčastěji pracují (je tedy možné zvolit více odpovědí). Výsledky jsou znázorněny na grafu 1 níže. Dominantou je tedy jednoznačně Git, který ze všech respondentů na danou otázku (74 298) zvolilo až 87 %. (STACK OVERFLOW, 2018)



Graf 1: *Používané verzovací systémy - stackoverflow.com průzkum*

2.2.3 TEXTOVÝ EDITOR A SDK

Je dobré umět ovládat na vysoké úrovni některý z dostupných textových editorů. I když samotné IDE obsahuje textový editor, někdy může být zbytečné ho pro některé typy úloh spouštět. IDE je vyvíjené jako celek pro vývoj, kdežto samotné textové editory cílí opravdu na operace s textem. Pro některé typy úloh tak mohou být o mnoho rychlejší, a to jak ve výkonu, tak v tom, jak rychle lze danou úlohu v editoru vyřešit. Jako příklad je možné si představit logovací soubory, které obsahují desítky tisíc řádků. Zacházení s takovým souborem může být jednodušší v samotném textovém editoru než v tom, které se nachází přímo v IDE. Mezi nejpokročilejší textové editory patří jednoznačně Vim[8].

Sada nástrojů pro vývoj se označuje jako SDK – Software Development Kit. SDK obsahuje sadu nástrojů, knihoven, dokumentaci, ukázky kódu, procesy a průvodce, díky kterým může vývojář vyvíjet pro danou platformu. (SANDOVAL, 2016) Pro vývoj v Javě nebo jiném jazyce využívající Java Virtual Machine (dále jako JVM), což je například Kotlin, je nutné mít stažené a nainstalované Java Development Kit, dále jen JDK, pro danou platformu (v době psaní této práce je dostupné JDK 10 pro většinu platforem).

2.3 SESTAVENÍ APLIKACE – BUILD

Zapsaný zdrojový kód umístěný ve verzovacím systému je samozřejmě zásadní, mnohem zásadnější je však to, aby byl převeden do nějakého pozorovatelného nebo hmatatelného

[8] Zdroj: (VIM, 2018).

výsledku. To se označuje jako „build“ (sestavení) aplikace. Zjednodušeně řečeno, dochází v rámci tohoto kroku k překladu lidsky čitelného kódu do spustitelné prezentace na dané platformě. V případě jazyka Java vznikne pro každou třídu v souboru *.java soubor *.class, jež bude obsahovat zkompilovaný kód – Java bytecode. Tuto část má na starost kompilátor, pro Javu nejčastěji javac. To je však opravdu základ a v praxi zahrnuje build mnohem více úloh. Je hlavně na nás, co do tohoto procesu zahrneme. Může se jednat například o kompilaci, testovaní, inspekci kódu, balíčkování a deploy nové verze. (DUVALL, 2007) Jednotlivé kroky lze realizovat jak manuálně, tak lze i využít některý z build nástrojů, který tyto kroky automatizuje a usnadňuje tak práci. Pro Javu je to například Maven nebo Gradle. Autor využívá ve své praktické části Maven, který je popsán v kapitole 4.2.6.

Je možné definovat různé buildy, tedy takové buildy, jež obsahují jiný sled kroků. Podle Duvalla (2007) je možné definovat následující třívrstvou hierarchii buildů:

1. privátní build,
2. integrační build,
3. release build.

2.3.1 PRIVÁTNÍ BUILD

Ještě před nahráním změn do VCS by si měl sám vývojář lokálně spustit build. Díky tomu, že je tento build spuštěn se změnami, které jsou lokálně integrovány s nejnovějšími změnami ve VCS, je tak možné odhalit nefunkční build. Postup by měl být následující:

1. stažení nejnovějšího kódu z verzovacího systému,
2. provedení úprav v kódu,
3. získání nejnovějšího kódu z verzovacího systému,
4. spuštění lokálního buildu zahrnujícího veškeré automatické testy,
5. nahrání změn z kroku 2. do verzovacího systému.

2.3.2 INTEGRAČNÍ BUILD

Nad všemi změnami, které vývojový tým provede a nahraje je do VCS, by měl proběhnout takzvaný integrační build. Pro tento typ buildu by měl být ideálně vyhrazen samostatný počítač. V rámci tohoto buildu je ještě možné provést rozklad na hlavní integrační build, který opravdu pouze testuje správnou integraci posledních změn a sekundární build. Hlavní build by neměl trvat déle než 10 minut. Do sekundárního buildu je možné zahrnout typicky pomalé úlohy, jako jsou výkonnostní testy nebo třeba automatickou inspekci kódu.

2.3.3 RELEASE BUILD

Výsledkem release buildu je taková verze softwaru, kterou je možné distribuovat uživatelům/zákazníkům. Typicky je tato verze vytvořena na konci nějaké časové iterace (v případě metodiky Scrum jím může být konec sprintu) a musí projít všemi akceptačními testy. Pokud si to povaha aplikace žádá, může být součástí i vytvoření instalačního média.

2.3.4 OZNAČOVÁNÍ VERZÍ

Kdo někdy nějaký software používal určitě, zaregistroval, že typicky za názvem aplikace nebo v nějaké části s informacemi o aplikaci jsou uvedena různá čísla. Jedná se o označení vydané verze aplikace (samozřejmě za předpokladu, že číslo nepatří do samotného názvu aplikace). Při označování verzí je doporučeno držet se všeobecně přijatých standardů. Nejrozšířenější je sémantické verzování (SemVer). V tomto verzování se zapisují verze následovně: MAJOR.MINOR.PATCH. Navyšování verzí se řídí následujícími pravidly: (TOM PRESTON-WERNER, 2018)

- MAJOR se inkrementuje, pokud nastala změna v kódu, která brání zpětné kompatibilitě (např. změna schéma databáze, změna v Application Programming Interface, dále API),
- MINOR se inkrementuje, pokud byla přidána nová funkcionalita, při které je zachována zpětná kompatibilita,
- PATCH se inkrementuje, pokud dojde k opravě chyby se zachováním zpětné kompatibility.

Výše zmíněné číslování se ještě často rozšiřuje pomocí přidaných metadat, například alfa verze označená jako 1.0.0-alfa, případně 1.0.0-beta apod. V rané fázi vývoje se využívá číslování 0.y.z, což značí, že v aplikaci dochází k častým změnám. První verze 1.0.0 by měla být vydána s veřejným API. Pokud již byl balíček aplikace pod nějakou verzí vydán, nikdy se do té samé verze nesmí provádět změny a je nutné vydat novou, vyšší verzi.

2.4 TESTOVÁNÍ APLIKACE

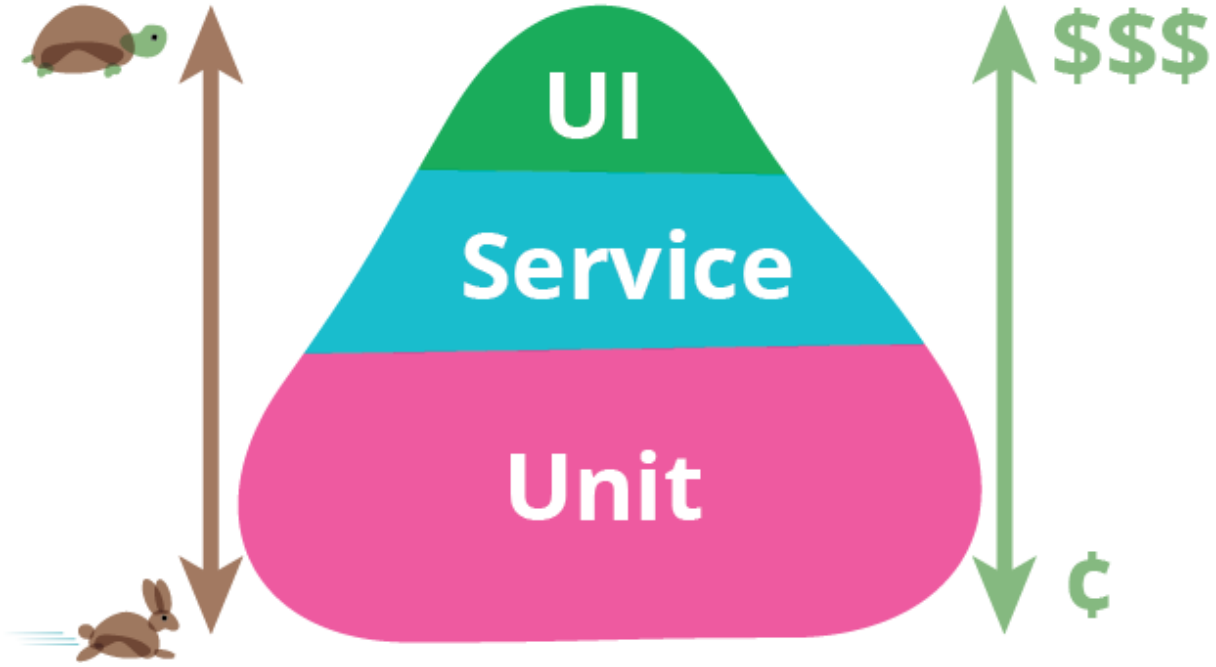
Až do této chvíle se mluvilo o tom, že testy jsou součástí buildu, že nám dávají zpětnou vazbu. To je sice pravda, ale jedná se pouze o jednu sadu testů. Těchto sad však existuje více. Je nutné si ale přesně definovat, co to testování softwaru je a jak můžeme testy rozdělovat. Existují totiž i testy, které jsou manuální.

Testování je proces identifikace správnosti, celistvosti a kvality vyvíjeného počítačového softwaru. (ONE STOP TESTING, 2018) Jinými slovy testování provádíme, abychom odhalili chyby a neočekávané chování programu a tyto nálezy mohli opravit. Tyto chyby jsou označovány slovem „bug“. Hodnocení kvality je i normalizované dle ISO/IEC 9126-1:2001 Software engineering – Product quality. Myers ve své knize zmiňuje tři důležité principy testování, na které je nutné vždy myslet: (MYERS, 2004)

1. testování je proces spouštění aplikace s úmyslem najít chyby,
2. dobrým testovacím případem je ten, který má vysokou pravděpodobnost nalezení dosud neidentifikované chyby,
3. úspěšný testovací případ je ten, který nalezne dosud neidentifikovanou chybu.

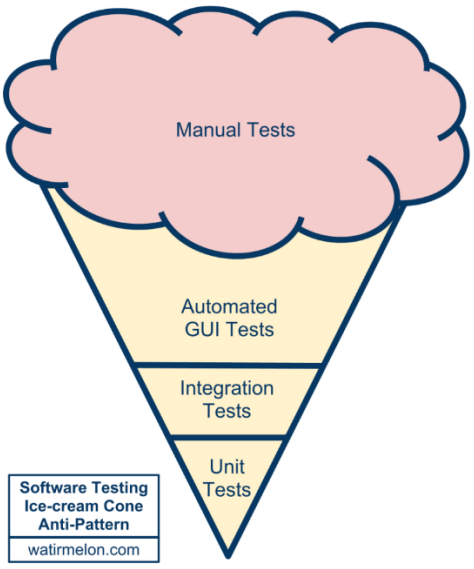
Rozdíl mezi automatizovanými testy a manuálními je jednoduchý a zřejmý z jejich označení. Méně znalý člověk této problematiky si nejspíše vždy představí manuální procházení aplikace a hledání chyb. Ano, i toto je jeden z přístupů a pro některé extrémní případy je nevyhnutelný. Avšak cílem je vždy snaha o automatické testování, které dokáže ušetřit čas a je možné stejný případ testovat rychle při každé změně. Nicméně testy lze dělit do několika skupin podle jejich určení. Kromě těch nejznámějších skupin, jež některé jsou popsány dále, existuje také mnoho dalších skupin. Vždyť jen Wikipedie, v rámci stránky věnované testování softwaru, uvádí až 19 skupin.

Testovací pyramida je jeden z ukazatelů důležitosti automatických testů. Na obrázku 4 je vidět, jak jsou tyto skupiny testů v rámci pyramidy rozděleny do částí, jež jsou různě velké. Toto rozdělení však není náhodné a má své opodstatnění. Nejvíce by měl být software pokryt jednotkovými testy (růžová část obrázku), které jsou nejlevnější a zároveň nejrychlejší. Prostřední část tvoří testy (modrá část obrázku), jež testují různé služby, například REST API. Těch by mělo být méně. Na vrcholu pyramidy jsou testy v rámci grafického uživatelského rozhraní, dále jako GUI, mezi které se řadí například automatické procházení webového prohlížeče pomocí Selenium frameworku. Jejich komplexita je řádově vyšší a rychlost nízká, takže se jedná o nákladné testy a mělo by jich v projektu být nejméně. (VOCKE, 2018)



Obrázek 4: *Testovací pyramida*[9]

Proti výše zmíněné testovací pyramidě stojí takzvaný zmrzlinový kornoutek, jež se řadí mezi špatné návrhové vzory. Jedná se vlastně o obrácený postup s tím, že nejvíce úsilí je věnováno do manuálního regresního testování, potom do GUI testů, do testů služeb a až nakonec do jednotkových testů. Tento špatný návrhový vzor je na obrázku 5.



Obrázek 5: *Ice-cream Cone Anti-Pattern*[10]

[9] Zdroj: (VOCKE, 2018).
[10] Zdroj: (ALISTER, 2018).

2.4.1 JEDNOTKOVÉ TESTY

Jednotkové testy (v angličtině Unit testy) jsou vykonány během fáze buildu aplikace. Tento typ testů je automatizovaný. Za jednotku je možné považovat testovatelnou kódovou část programu, například třídu nebo metodu. Během těchto testů typicky není použita integrace s databází nebo jiným http serverem. Zpravidla každá nová část kódu by měla být pokryta jednotkovým testem. Tento druh testů dává nejrychlejší odezvu, zdali se zásahem do nějaké části kódu neovlivnila jiná část. (BECK, 2002) Tyto testy využívají různé testovací frameworky, nejznámější z nich pro Javu je JUnit, aktuálně ve verzi 5. Pro .NET je pak známý NUnit. Jednoduchý příklad testované metody a jednotkového testu je zde:

Zdrojový kód 1: *Třída GreetingMachine obsahující metodu greet*

```
public class GreetingMachine {
    public String greet(String name){
        return "Hello " + name;
    }
}
```

Zdrojový kód 2: *Třída GreetingMachineTest obsahující metodu testGreet*

```
import org.junit.Test;
import static org.junit.Assert.*;

public class GreetingMachineTest {
    @Test
    public void testGreet() {
        GreetingMachine gMachine = new GreetingMachine();
        String result = gMachine.greet("Pavel");
        assertEquals("Hello Pavel", result);
    }
}
```

Ve zdrojovém kódu 1 je metoda „greet“ s jedním vstupním parametrem typu String ve třídě „GreetingMachine“, která vrací objekt typu String, jež je složen z pozdravu „Hello“ a následně hodnotou obdrženou parametrem. Jednoduchý jednotkový test, jak je vidět ve zdrojovém kódu 2, vytvoří instanci třídy „GreetingMachine“, nad touto instancí zavolá metodu s parametrem „Pavel“ a následně se ověřuje, jestli výsledek splňuje očekávaní, tedy jestli se

hodnota rovná „Hello Pavel“. Výstupem takového testu je jeho označení buď jako úspěšný nebo neúspěšný.

2.4.2 INTEGRAČNÍ TESTY

Poté, co úspěšně proběhne build aplikace, jsou nad výsledným sestavením vykonávány integrační testy. V rámci těchto testů se testuje integrace s kritickými částmi, jako například s databází, http serverem nebo s jinými komponenty. Těchto testů je zpravidla menší počet než jednotkových. Integrační testy mohou být automatizované nebo manuální. O jejich vytváření se stále stará dodavatel, tedy jeho testovací team, případně i vývojáři. Typicky je snaha otestovat všechny případy použití, a to i ty velmi okrajové. (SMARTBEAR SOFTWARE, 2018b) Pro integrační testy se využívají různé nástroje a frameworky. V rámci této diplomové práce a její praktické části autor pro integrační testování využívá Geb[11] pro automatizaci webového prohlížeče spolu s testovacím frameworkem Spock[12]. Příklad testovacího scénáře je uveden v tabulce 2. Jedná se o ukázku možného integračního testu přímo nad vyvíjenou aplikací z praktické části této práce. Zároveň je ve zdrojovém kódu 3, reálná implementace tohoto scénáře využívající právě Geb i Spock. Definice obsahu webové stránky, obsahující výpis článků, je obsažena ve třídě „ZArticlesPage“.

Tabulka 2: *Ukázkový testovací scénář*

Název testu	Změna počtu vypisovaných článků na stránku	
Krok	**Popis kroku**	**Očekávaný výsledek**
1.	Otevřít web Zoumi.cz a přejít na stránku „Články“.	Adresní řádek obsahuje URL https://www.zoumi.cz/articles a na stránce jsou vypsané články s možností filtrování.
2.	Z výběru „Záznamů na stránku:“ vybrat hodnotu 10.	Stránka se obnoví a bude obsahovat 10 článků.

[11] Geb je řešení pro automatizaci procházení webového prohlížeče. Využívá programovací jazyk Groovy. (GEBISH, 2018)
[12] Spock je framework pro testování Java a Groovy aplikací. (NIEDERWIESER, 2018)

Zdrojový kód 3: *Implementace testovacího scénáře*

```
def 'filtering contains page size drop down and when changed, it
does change value'() {
    given:
    to ZArticlesPage

    when: 'select 10 from dropdown...'
    dropdownPageSize = '10'

    then:
    dropdownPageSizeSelectedText == '10'
    dropdownPageSize.value() == '10'
}
```

2.4.3 SMOKE TESTY

Ještě před samotným započetím systémového testování je nutné ověřit, že kritické oblasti aplikace fungují správně a má tak smysl, aby tester začal testovat další testovací případy. Může se jednat například o ověření, že aplikace byla správně sestavena nebo je správně nasazena (je správně nastaveno produkční prostředí). Nejedná se tedy o vyčerpávající testování, ale spíše o takovou prvotní jistotu, která ukáže, zdali je software použitelný pro další fáze testování. (SHARMA, 2016)

2.4.4 AKCEPTAČNÍ TESTOVÁNÍ

Po úspěšném otestování softwaru na straně dodavatele dochází k jeho předání a nasazení do zákazníkova prostředí. V tuto chvíli zákazník, respektive jeho testovací tým, provede vykonání takzvaných „User Acceptance Test“ - UAT. To jsou testy, které si definuje přímo zákazník, případně ve spolupráci s dodavatelem. Dle Hlavy zákazník očekává, že v systému nějaká chyba bude a je spokojen, pokud jeho tým chybu najde. Ovšem pokud se jedná o mnoho chyb, je zákazník nespokojen. Pokud zákazník chybu objeví, předá informace dodavateli, který v co nejkratší době chybu opraví a dodá opravený software, aby bylo možné aplikaci dále testovat. (HLAVA, 2011)

Je velice důležité mít tento typ testů přesně zanesen ve smlouvě. Tento typ akceptačního testování se nazývá „Contract Acceptance Testing“ a je definován smlouvou, kterou podepisují obě strany – dodavatel a kupující. (ORIGINAL SOFTWARE, 2018)

3 CONTINUOUS INTEGRATION

V předcházejících kapitolách byl předveden velice základní přehled o tom, jak se postupuje v rámci vývoje softwaru, jaké se používají nástroje a také zmínka o tom, že existují různé druhy testů. Je nutné mít o této problematice přehled pro správné pochopení metodik kontinuální integrace. CI zasahuje nejen do toho, jak pracuje tým nebo jak je definovaný build, ale také vyžaduje automatické testy, přístup k verzovacímu systému, a kromě toho hlavně server, který CI podporuje.

3.1 CO JE TO CONTINUOUS INTEGRATION?

Co to vlastně kontinuální integrace znamená? Nejvýraznější osoba této domény, Fowler, ve svém článku uvádí následující: (FOWLER, 2006)

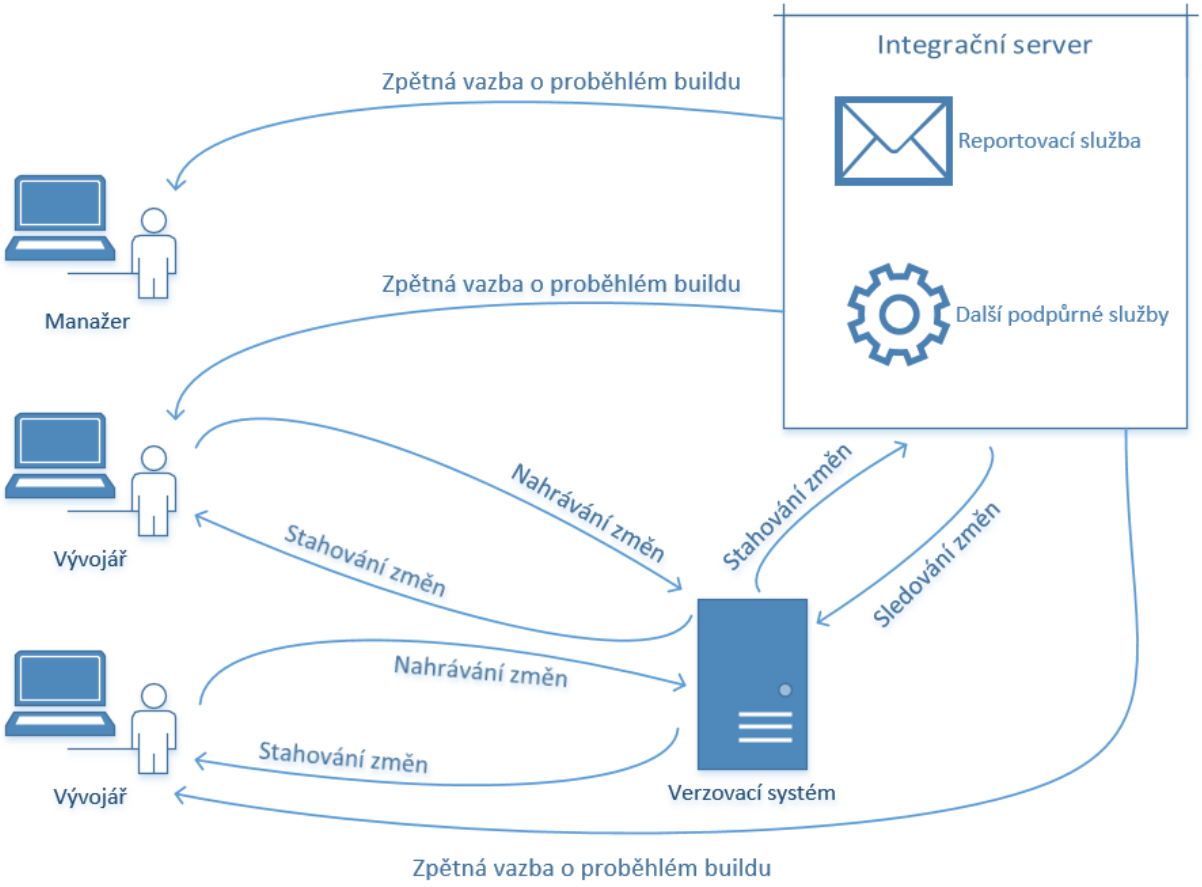
Kontinuální integrace je praktika vývoje softwaru, kdy členové vývojového týmu integrují jejich práci často, nejčastěji každý člen alespoň jednou denně. To vede k několika integracím za den. Každá nová integrace je ověřena automatickým buildem s testy, což vede k odhalení případných integračních problémů co nejdříve.[13]

Z předchozích kapitol je již zřejmé, jak probíhá vývoj podle různých metodik a jaké nástroje se používají. Před detailnějším popisem jednotlivých částí následuje jednoduchý příklad adaptace CI. Tým několika lidí, skládajících se z vývojářů a testerů, pracuje společně na vývoji produktu. Proces kontinuální integrace začíná u vývojáře, který tvoří kód vyvíjeného programu v některém z IDE a své změny nahrává do VCS. Ten je dostupný všem ostatním vývojářům (ať už pouze v rámci týmu, nebo otevřený a dostupný online), a ti do něj nahrávají i své změny. V ideálním případě vývojář provádí menší změny a ty vždy po dokončení (a po úspěšném lokálním buildu) nahraje do VCS. Na tento verzovací systém je připojen některý z CI serveru, který nad posledními změnami spustí definovaný build. Tento build může být buď úspěšný, v takovém případě se jedná o chtěný stav, nebo odhalí chyby, které vznikly. V případě chybného buildu by měl celý tým i jiné zainteresované osoby obdržet takovou informaci co nejdříve a veškeré své síly věnovat na zjištění příčiny chyby a poté její nápravě.

Výše uvedený příklad je opravdu velice zjednodušený, ale nastiňuje základní princip CI tak, jak ho popsal Fowler. Každá část cyklu má své vlastní úskalí a je třeba si jí jasně definovat

[13] Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.

a přizpůsobit povaze aplikace. Také se často nejedná pouze o jediný build obsahující všechny kategorie testů nebo inspekci kódu, ale tyto buildy se rozdělují do menších. Na obrázku 6 je znázorněna typická architektura při používání CI.



Obrázek 6: *Schéma prostředí využívající CI*

3.1.1 VÝVOJÁŘ

Ve chvíli, kdy dokončí vývojář práci na dané úloze, musí tyto změny nahrát do VCS. Ještě před tím by si však měl aktualizovat zdrojové soubory na nejnovější z VCS a provést svůj privátní build (viz kapitola 2.3.1). Tento build by měl obsahovat kompilaci zdrojových kódů a jednotkové testy. V případě úspěchu nic nebrání nahrání do VCS. Pokud se během lokálního buildu projeví chyba (ať už kompilační nebo chybný test), je nutné analyzovat problém a provést nápravu. Do VCS nikdy nenahráváme nefunkční kód. Build i operace s VCS může vývojář provádět z IDE i z příkazové řádky, vždy záleží na použitých nástrojích a jejich možnostech.

3.1.2 VERZOVACÍ SYSTÉM

Verzovací systém (VCS) a to, k čemu slouží, bylo již popsáno v kapitole 2.2.2. V této části je nutné popsat to, jak musí být nastaven pro správnou funkcionalitu CI. Verzovací systém může být umístěn přímo v rámci firemní sítě jako on-premise[14] řešení, může být nainstalovaný v rámci PaaS (např. AWS[15]) nebo je možné využít některou z online hostovaných služeb, jako například GitHub, GitLab, BitBucket a mnoho dalších. Všechny mají společné to, že slouží k uložení kódu, avšak některé mohou poskytovat služby navíc, a to třeba právě i nástroje pro CI. V rámci bezpečnosti kódu lze také mluvit o rozdělení na veřejné a privátní VCS. Autor práce uvádí hlavně rozdělení na veřejný a privátní VCS z důvodu, aby ukázal, že pro obě platí trochu odlišné přístupy při nasazování CI. Každý jednotlivec, tým i organizace se musí sami rozhodnout, jakou cestou jít. Je však důležité, že v každém případě existuje řešení, jak CI nasadit.

3.1.2.1 Veřejný verzovací systém

Veřejný VCS, ať už v rámci on-premise řešení nebo pomocí některé z online služeb, představuje jedno z nejjednodušších řešení při nasazování CI serveru. Jedna z prvotních úloh CI serveru je monitorovat změny ve VCS a jejich stažení. V tomto případě je tak možné využít většiny z online služeb (Travis CI, Circle CI a další) pro CI nebo spravovat vlastní instalaci CI serveru lokálně (Jenkins, TeamCity a další), protože na VCS vždy (za předpokladu funkčního internetového připojení) dosáhnou. V takovém případě je ale nutné si položit otázku, jestli je obsah zdrojových souborů natolik bezpečný, případně právně ošetřený, aby mohl být publikován veřejně.

3.1.2.2 Privátní verzovací systém

Pokud je VCS umístěn na některé z online služeb a je pouze označen jako privátní, pak je možné využít takový CI server, který podporuje nějakou formu autentizace[16] a bude schopný se k VCS přihlásit. V případě, že se jedná o on-premise řešení umístěné v lokální síti a server není (například z bezpečnostních důvodů) dostupný na vnější síti, je nutné využít takový CI server, který lze instalovat a provozovat lokálně. Takovým příkladem může být Jenkins, který je popsán v rámci kapitoly 3.2.2.

[14] Společnost spravuje hardware i software sama.
[15] AWS – Amazon Web Services je cloudová platforma, poskytující výpočetní výkon, databázové úložiště a další služby pro podporu růstu bussinesu. (AMAZON, 2018)
[16] Proces ověřování proklamované identity. (TURNER, 2016)

3.2 CI SERVER

Jedná se o nástroj, díky kterému je možné provádět kontinuální integraci. CI server nejčastěji monitoruje VCS a v případě změny stáhne nejnovější zdrojové soubory a spustí sestavení spolu s jednotkovými a integračními testy. Výsledkem je takový artefakt aplikace, který lze dále testovat, případně rovnou nasadit do produkčního prostředí (to je označováno jako „Continuous Deployment“). CI server informuje tým o úspěšném nebo neúspěšném sestavení některým z komunikačních kanálů (email, SMS, Slack[17] a další). (THOUGHTWORKS, 2018) Samozřejmě každý CI server je trochu jiný a poskytuje více, nebo naopak méně dalších funkcí. Primárně ale všechny dokážou kontinuálně sestavovat aplikaci.

3.2.1 HUDSON

Jedna z prvních nejoblíbenějších alternativ Cruise Controlu vyšla v roce 2005 a nese název Hudson. Nástroj primárně vyvíjel Kohsuke Kawaguchi, zaměstnanec Sun Microsystems, a vycházel jako svobodný software. V roce 2008 se stává dokonce oblíbenější než CruiseControl a vyhrává ocenění „Duke’s Choice Award“ v kategorii „Developer Solutions“. Poté, co byl Sun koupen společností Oracle, jež si nárokovala název Hudson jako ochrannou značku, vznikl nový projekt, který nese označení „Jenkins“. Tento název byl drtivou většinou komunity schválen při hlasování 29. ledna 2011.

Nástroj je napsaný v jazyce Java a je možné ho tedy spustit na jakémkoliv aplikačním serveru (Apache Tomcat, Payara a další). Podporuje většinu verzovacích systémů a mimo podpory Apache Ant a Apache Maven podporuje také vykonávání shell scriptů a příkazů z Windows. Je možné jej také rozšířit pomocí různých pluginů. (KAWAGUCHI, 2007)

3.2.2 JENKINS

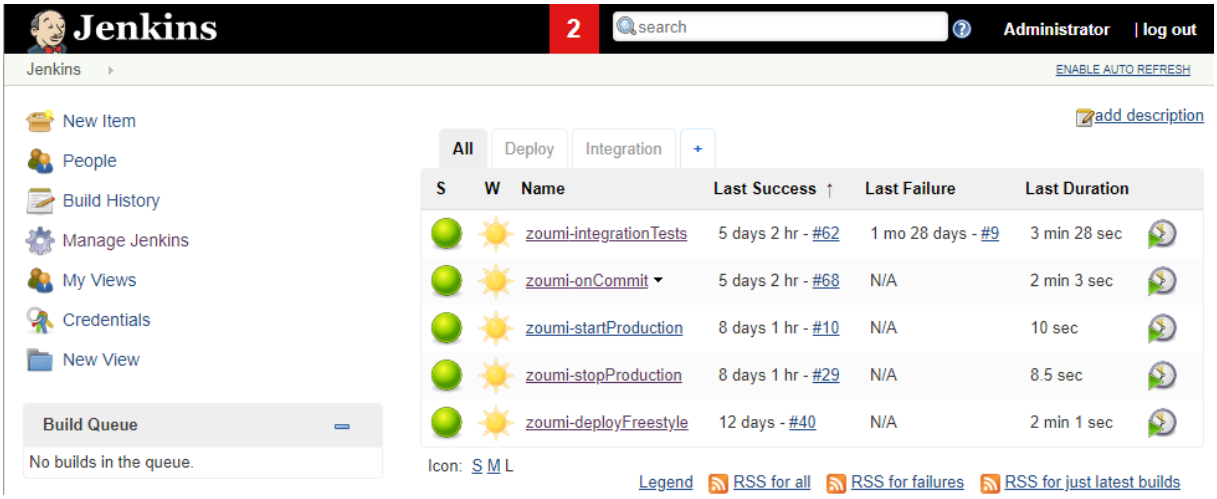
Tento nástroj vznikl z důvodu, který je uveden v předešlé kapitole. Jenkins je open source Java aplikace, která slouží k automatizaci nejrůznějších úloh, týkajících se sestavení, testování, dodávání a nasazování počítačového softwaru. Pro svůj běh potřebuje nainstalovanou Javu. Je distribuován pomocí nativních systémových balíčků, dále také jako Docker image případně je možné ho stáhnout jako balíček a spustit kdekoliv. (JENKINS, 2018) Instalace je popsána v rámci kapitoly 4.5.1.

[17] Slack je nejen komunikační nástroj podporující práci v prostředí s více lidmi. (SLACK, 2018)

Požadavky na hardware nejsou přesně dané a závisí na několika faktorech a správném návrhu architektury. Minimálně by však měl stroj, kde bude Jenkins nainstalovaný, disponovat alespoň 256 MB operační paměti a 1 GB volného místa na disku. Jenkins totiž dokáže fungovat jako model master-agent, kdy master je centrální bod, který pro vykonání sestavení využívá jednotlivé dostupné agenty. Úkolem master serveru je odbavování http požadavků a ukládání důležitých informací (konfigurace, historie sestavení a rozšíření) do složky, která je dostupná pod $JENKINS_HOME. (JENKINS, 2018)

Zároveň je ale stále možné využít i master pro spouštění sestavení. Takto postavenou architekturu využívá i autor pro vývoj aplikace Zoumi.cz. Zároveň ze své zkušenosti z praxe konstatuje, že rozdělení sestavení na více agentů je vhodné v případě rozsáhle aplikace.

Centrálním místem při využívaní Jenkinse je „Dashboard“, který je vidět na obrázku 7. Ten lze přizpůsobit, ale v základním nastavení obsahuje přehledný výpis všech nastavených jobů s důležitými informacemi jako je status posledního sestavení, znázorněn pomocí barevné koule (sloupec S). Dále ukazatel stability (sloupec W), jež je agregován z několika posledních sestavení. Nechybí samozřejmě časové informace, jako kdy proběhlo poslední sestavení, kdy naposled vznikl při sestavení problém a také jak dlouho sestavení trvalo.



Obrázek 7: *Zoumi.cz - Jenkins dashboard*

Job

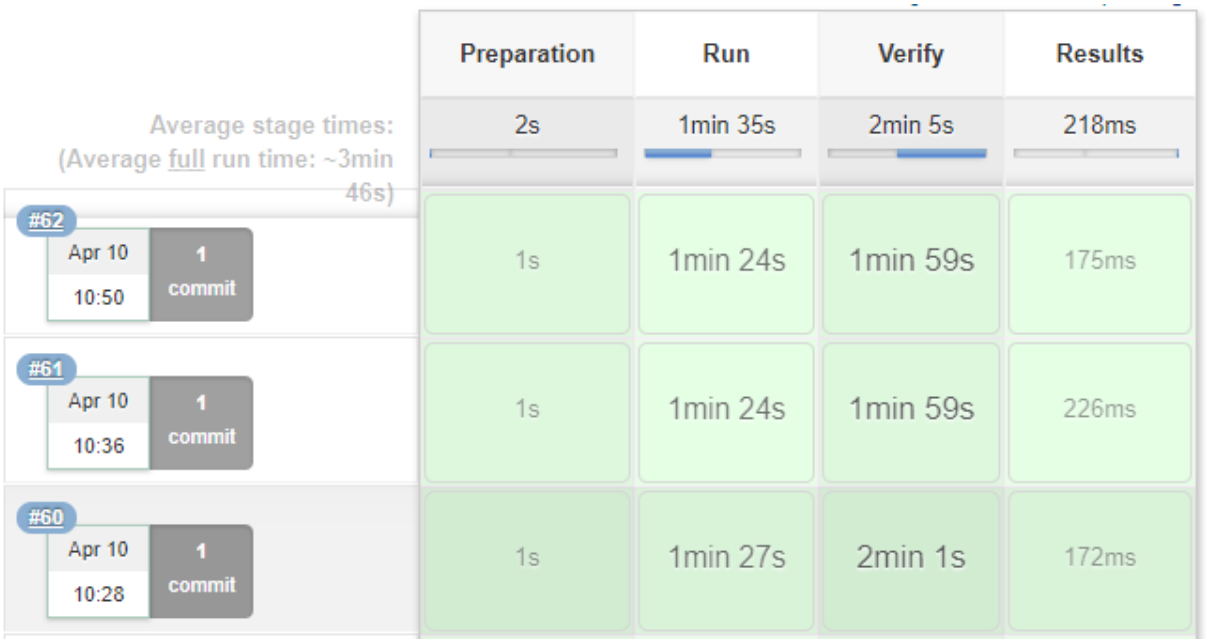
Job je základní prvek, který definuje samotné sestavení. To je to místo, kde je možné zvolit, jaký VCS se bude sledovat, jak často se bude sledovat, jaké kroky se vykonají před, během a po sestavení. To samozřejmě není z nastavení vše, ale jedná se o to základní. Každý takto definovaný job je pak na základě definovaných podnětů spouštěn a k těmto spuštěním je vedena

historie (počet sestavení i dobu ukládání historie lze také nastavit). Při vytváření nového jobu si lze vybrat šablonu, která projektu nejvíce vyhovuje. Mezi základní patří:

- freestyle project,
- pipeline.

Freestyle project je základní typ definice jobu, který poskytuje dostatečnou volnost v různém nastavování. Nemusí být ani přímo využíván pro sestavování aplikace, ale i pro jiné typy úloh. Veškeré nastavování probíhá formou výběru, případně vyplňováním hodnot ve webovém rozhraní aplikace.

Pipeline byly do Jenkinse přidány teprve v roce 2016 s příchodem verze 2.0. (KAWAGUCHI, 2016) Díky nim by mělo být ještě jednodušší do projektu nasadit také „Continuous Delivery“. Mezi další výhody patří například definice projektu pomocí kódu a možnost sdílení pipeline mezi týmy. Další přidaná vlastnost je i vylepšená vizualizace během vykonávání pipeline přímo v detailu jobu tak, jak je vidět na obrázku 8.



Obrázek 8: *Fáze sestavení aplikace v rámci pipeline jobu*

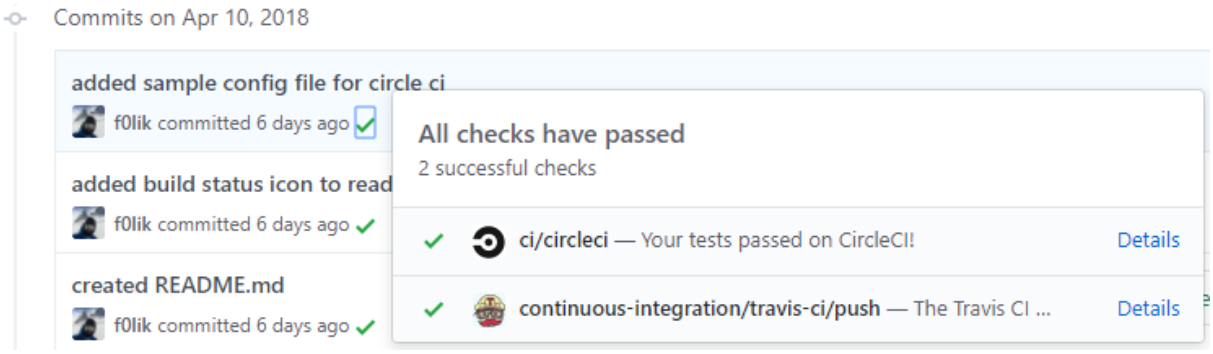
Jenkins se jednoznačně řadí mezi silné nástroje a je možné si ho kompletně přizpůsobit. Dále podporuje i pluginy, takže v případě nedostupnosti některé funkcionality přímo v základní instalaci je dost velká šance, že bude existovat plugin. A pokud ani to ne, není problém si vytvořit vlastní. Autor práce využívá Jenkins jako primární CI server pro vývoj aplikace Zoumi.cz. Instalace je poměrně jednoduchá, oproti následujícím službám však vyžaduje podstatně více úsilí pro uvedení do použitelného stavu.

3.2.3 TRAVIS CI

Travis CI je webová služba poskytující CI pro projekty, které jsou hostovány na GitHubu. Pro projekty vyvíjené jako open source je možné službu využívat zdarma. Za různý poplatek, pohybující se od 69 dolarů do 489 dolarů za měsíc[18], je pak možné využívat službu i pro ostatní projekty. Nastavení služby je velice jednoduché a provede se přidáním souboru „.travis.yml“ do kořenové složky projektu. Tento soubor obsahuje hlavně definici programovacího jazyka, požadované kroky sestavení a nastavení testovacího prostředí. Po autorizaci služby pro daný repozitář GitHub oznámí při každém novém commitu nebo pull requestu tuto událost a na Travis CI vzniká nový build. Výsledky buildu je možné oznámit pomocí různých kanálů, jejichž nastavení je možné provést pomocí konfiguračního souboru. Travis CI podporuje sestavení softwaru pro několik jazyků, mezi něž patří mimo jiné C, C++, C#, Java, PHP, Ruby, Scala a další. Celkem podporuje 31 programovacích jazyků. (TRAVIS, 2018b) Po každém sestavení je dostupný informační banner, obrázek 9, o stavu buildu, který lze umístit například přímo k popisu repozitáře na GitHubu a dát tak všem informaci o tom, v jakém stavu se nachází kód po poslední změně.

Obrázek 9: *Ukazatel stavu sestavení na Travis CI*

Mezi další výhodu patří také to, že díky přímému napojení na GitHub účet je ke každému commitu přidána informace o stavu buildu. To se například v Jenkinsu musí složitě nastavovat. Jak takové označení vypadá, je vidět na obrázku 10, který také obsahuje informaci o stavu sestavení z další služby, jež je popsána v následující kapitole.



Obrázek 10: *Označení stavu sestavení po provedeném commitu na GitHubu*

[18] Hodnoty ke dni 16. 4. 2018. (TRAVIS CI, 2018a)

Autor práce v rámci praktické časti otestoval i tuto službu a od zaregistrování, nastavení až po spuštění prvního buildu uběhlo jen několik jednotek minut. Samozřejmě velice záleží na rozsahu a povaze aplikace. Build aplikace společně s unit testy trval přesně 1 minutu a 39 sekund. Kromě kompletního výpisu z konzole obsahuje webové rozhraní jen několik základních informací o průběhu buildu.

3.2.4 CIRCLE CI

Circle CI je další z webových služeb pro podporu CI. Stejně jako předchozí dokáže pracovat s repozitáři na GitHubu a přidává podporu pro BitBucket (pouze ve verzi Cloud). V rámci bezplatné varianty má vývojář k dispozici jeden linuxový kontejner bez paralelizace, který může využívat pro sestavení až 1500 minut za měsíc. S narůstajícím počtem kontejnerů, paralelizace, případně počtem volných minut za měsíc, přichází i poplatky, pohybující se v řádu několika desítek dolarů až jednotek tisíců dolarů za měsíc. To vše je možné využívat jako cloudovou aplikaci. Pro firmy vyžadující vysoké zabezpečení je dostupná verze, kterou si mohou nainstalovat a spravovat sami ve svém prostředí. Konfigurace služby je součástí souboru config.yml, jež je umístěn v kořenovém adresáři projektu, konkrétně ve složce „.circle.ci“. Takový soubor obsahuje na prvním řádku verzi Circle CI, dále Docker image pro daný zvolený jazyk, a také rozdělení na fáze build, test a deploy, přičemž pro každou fázi lze definovat požadované kroky. Díky integraci s GitHubem dokáže Circle CI monitorovat změny a na jejich základě spustit nové sestavení a po jeho dokončení, ať už úspěšném nebo ne, zaslat notifikační email. Lze také nastavit integraci s nástroji jako je Slack, HipChat, Campfire, Flowdock a IRC notifikace. Stejně jako předchozí služba také dokáže označit commit na GitHubu se stavem proběhlého buildu, jak je vidět na obrázku 10. Mezi velké hráče na trhu, kteří Circle CI používají, se řadí Facebook, Spotify, Kickstarter nebo GoPro. (CIRCLECI, 2018)

Vůči předchozí popsané službě je rozhodně výhodou podpora BitBucketu a hlavně možnost využití bezplatné verze i pro komerční použití. Na druhou stranu podporuje méně programovacích jazyků, a to konkrétně pouze 9! Mezi ty se řadí Java, Clojure nebo Ruby.

Stejně jako u Travis CI i zde bylo nasazení aplikace velice jednoduché a rychlé. Stejný build aplikace zde však zabral podstatně méně času, a to přesně 57 sekund. Circle CI poskytuje k buildu více informací. Tento konkrétní build proběhl na přidělené konfiguraci se 2 procesory a 4096 MB operační paměti. To vše v rámci bezplatné verze. Mezi další informace patří například označení testu, který trval nejdelší dobu.

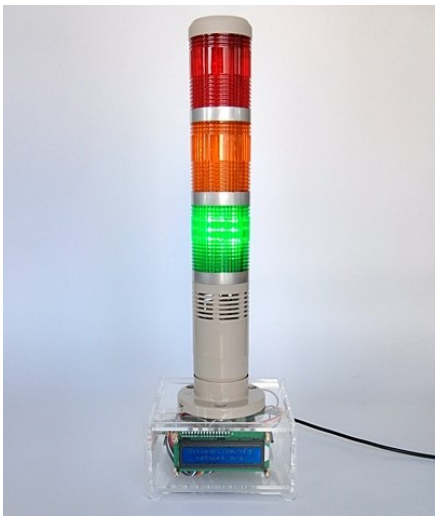
3.3 ZPĚTNÁ VAZBA

Nedílnou součástí CI je zpětná vazba neboli kontinuální reporty. Po každém proběhlém buildu disponuje CI server řadou cenných informací, nejdůležitější je samozřejmě stav sestavení. Jak s takovými informacemi nakládat? Je nutné vybrat vždy ty správné informace, ty posílat správným lidem ve správný čas a pomocí správného komunikačního kanálu.

Mezi základní informace se řadí stav sestavení, výsledek inspekce kódu, výsledky testů a případně informace o výsledku nasazení aplikace. Všechny informace není nutné zasílat kontinuálně, ale je možné nastavit plánovač, který v rozumném intervalu tyto informace odešle. Zpravidla každá role vyžaduje trochu jiné informace, a to také v různý čas. Je nutné si uvědomit, že pokud budeme zasílat všechny informace všem, povede to k ignoraci těchto zpráv. Projektový manažer vyžaduje hlavně feedback v reálném čase, takže často využívá „dashboard". Architekti běžně potřebují disponovat informacemi o stavu sestavení, protože na software nahlíží jako na celek. Vývojáři typicky potřebují informaci o kódu, který změnili a také výsledky testů a inspekce kódu nejnovějšího sestavení. Testery nejvíce zajímají výsledky automatických testů a inspekce kódu. Důležité je také se rozhodnout, kdy je správný čas pro zasílání informací. Díky možnostem CI serveru je možné definovat skupiny a různá pravidla, kdy proběhne zasílání a jaké informace bude obsahovat. Stále ale platí, že většinu informací chceme vědět hned. Mezi základní komunikační kanály se řadí email, SMS, rozšíření do prohlížeče, veřejně vystavený monitor. (DUVALL, 2007) Moderní CI servery disponují řadou dalších možností, jak informace doručit.

Někteří kutilové zacházejí až do takových extrémů, jako je sestavení vlastního fyzického indikátoru, nejčastěji ve formě lampy s barevnými diodami, jež indikují stav sestavení. Taková lampa je vidět na obrázku 11. Tato konkrétní se dá koupit již sestavená a stojí 279 €[19].

[19] Dne 18. 4. 2018 je to v přepočtu okolo 7100 Kč.



Obrázek 11: *Jenkins XFD Lamp*[20]

3.4 CONTINUOUS DELIVERY A CONTINUOUS DEPLOYMENT

Díky využívání CI jsou také otevřeny dveře pro další fáze, jež spolu úzce souvisí. Těmi jsou Continuous Delivery a Continous Deployment.

3.4.1 CONTINUOUS DELIVERY

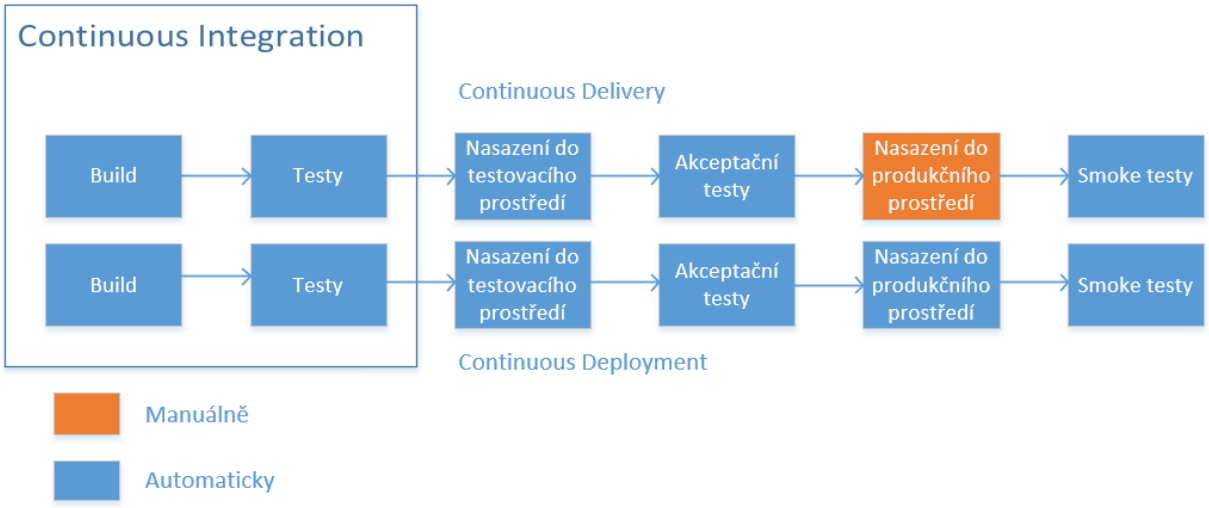
Jedná se o praktiku, díky které je možné vydávat nové fungující verze softwaru několikrát denně. Je tedy nutné stále udržovat aplikaci ve stavu schopném pro vydání. Pokud v rámci CI máme automatizovaný celý proces integrace aplikace, v rámci této praktiky je nutné mít automatizovaný proces vydání nové verze. Vydání nové verze je však ale stále spuštěno manuálně, například kliknutím na tlačítko. Pro aplikaci této praktiky je nutné mít nejen silný základ v CI, ale také velký počet testů pokrývající kód. Výhodou je právě zmíněný proces automatizace vydávání nové verze. (PITTET, 2018)

3.4.2 CONTINUOUS DEPLOYMENT

V rámci této praktiky se zachází ještě o trochu dále. Při každé změně, která úspěšně projde všemi fázemi zvolené pipeline, je aplikace automaticky nasazena do produkčního prostředí. Nejlépe je vidět rozdíl na obrázku 12. Kritické pro úspěch aplikace této praktiky jsou vysoce kvalitní testy. Dále je potřeba neustále přizpůsobovat dokumentaci softwaru k posledním změnám. Výhodou je rychlý vývoj a nasazení do produkčního prostředí. Není totiž nutné, aby změna byla nějakou dobu nepublikovaná. Vzhledem k častému nasazování změn je také vydávání nových verzí méně riskantní, protože obsahují méně změn. Pro zákazníky je přínosné,

[20] Zdroj: (GITGEAR, 2018).

že vidí neustálý tok nových vlastností místo toho, aby na verzi čekali měsíce či roky. (PITTET, 2018)



Obrázek 12: *Porovnání fází Continuous Delivery a Continuous Deployment*

4 WEBOVÁ APLIKACE ANALÝZA TEXTU

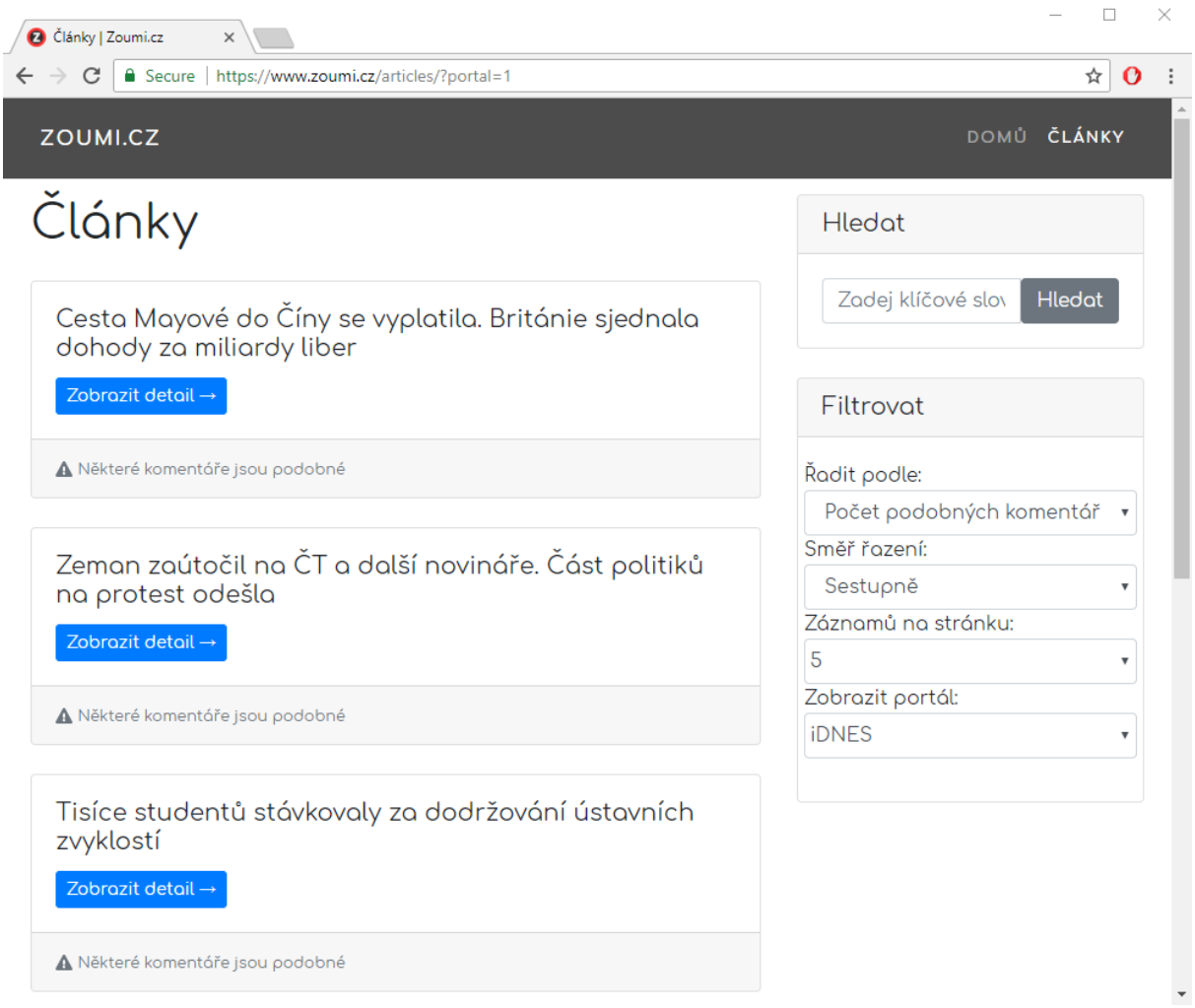
V rámci praktické části se autor zabývá vývojem webové aplikace v jazyce Kotlin, při jejímž vývoji plně využívá a aplikuje metodiky CI. Používaný CI server Jenkins si autor sám hostuje na svém serveru, stejně tak jako webovou aplikaci. K hostování je využit počítač o velikosti kreditní karty a je jím Raspberry Pi model 3. Komunikace s webovým rozhraním CI serveru i s aplikací je šifrována s využitím certifikátu od certifikační autority Let's Encrypt. Aplikace je vyvíjena a publikována iterativně, přičemž zdrojový kód je volně dostupný na portálu GitHub v autorově repozitáři.[21]

4.1 TÉMA WEBOVÉ APLIKACE ZOUMI.CZ

V dnešní době je internet masivně rozšířen a je také zdrojem informací, tak jako dříve jiná média. Výhodou internetu je, že na něm lze vystupovat anonymně a zveřejňovat téměř cokoliv. V případě konzumace informací z televize, rozhlasu nebo novin se jedná o jednosměrnou komunikaci ze směru od vydavatele ke konzumentovi, který si sám může vytvořit svůj vlastní názor a v tu samou chvíli diskutovat získané informace osobně se svým okolím. Jiný případ nastává v případě internetových zpravodajských portálů. Zpravodajské portály publikují na internetu články na různá témata a většina těchto portálů dovoluje vést uživatelům diskuzi pro každý článek. Tyto diskuze jsou semeništěm různých názorů ostatních uživatelů internetu, ale také mohou být prostorem pro snahu ovlivnit názor ostatních diskutujících, v lepším případě pak jen místem pro internetové trolly[22]. Autor této práce disponuje databází, která obsahuje strukturovaná data ve formě portál-články-komentáře. Data pocházejí z vybraných českých zpravodajských online portálů. Každý den probíhá sběr dalších dat, takže databáze se stále rozrůstá. Nutno dodat, že sběrem těchto dat se tato aplikace nezabývá, pouze již takto sesbíraná data přebírá do své databáze. A proč toto všechno? Cílem této práce je provést automatickou analýzu všech komentářů pro každý článek, zdali některé komentáře nejsou podobné, až téměř stejné. Výsledkem této analýzy je prezentace výsledků v rámci webové aplikace, která kromě základních statistik o počtu podobných komentářů poskytuje i možnost prohlížet jednotlivé články a jejich statistiky. Mezi články je možné vyhledávat pomocí klíčových slov a používat filtrování, jak je vidět na obrázku 13. Autor aplikace se za žádnou cenu nesnaží označit vadné komentáře, to je problematika, která není obsahem této diplomové práce. Zároveň se také jedná

[21] Zdroj: (GITHUB, 2018a).
[22] Jedná se o člověka, který se snaží svými komentáři na internetu poškodit co nejvíce lidí. Vybírá si nejčastěji kontroverzní témata a baví se tím, jak ostatním lidem svými komentáři škodí. (IN FLOW, 2008)

o surová data, tak jak je jiný systém získal a není nad nimi provedeno očištění, takže se zde mohou nacházet duplicity a ty jsou poté vyhodnoceny jako podobné. Aplikace by však měla být jakýsi indikátor pro čtenáře, aby si dával větší pozor na to, co čte. Pro pojmenování aplikace použil autor název Zoumi.cz (to je i internetová doména, kde je aplikace dostupná), kdy se jedná o zkrácení výrazu „Lžou mi?“.



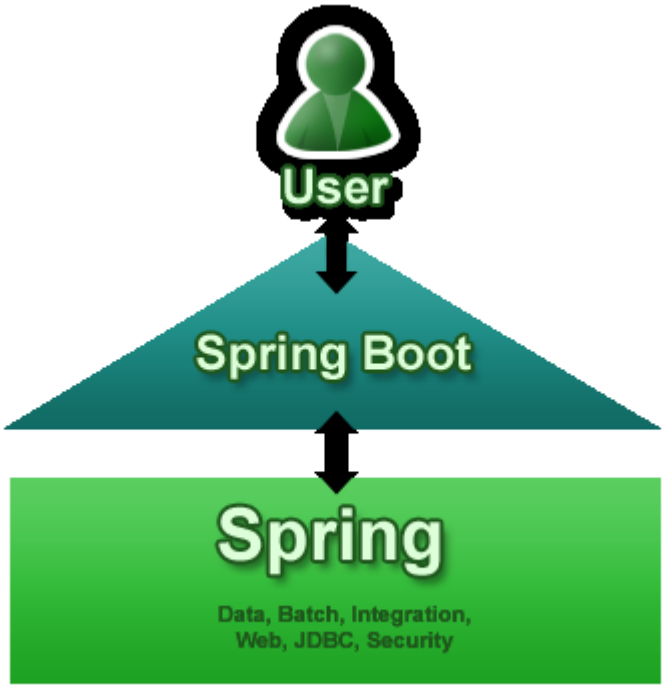
Obrázek 13: *Výpis článků v aplikaci Zoumi.cz*

4.2 POUŽITÉ TECHNOLOGIE

Při vývoji aplikace bylo použito několik různých technologií, ať už se jedná o nástroje nebo jazyk. Důležité je zmínit, že stejná aplikace by se dala sestavit i pomocí jiných technologií. Autor použil právě níže zmíněné, protože se s nimi chtěl naučit, nebo je již třeba znal, a proto si je vybral.

4.2.1 SPRING BOOT

Celá aplikace je postavena pomocí Spring Bootu. Jedná se o sadu předkonfigurovaných a již nastavených frameworků nebo služeb, díky kterým je možné za krátkou dobou, téměř bez žádného nastavování, spustit vlastní webovou aplikaci tak zvaně „out-of-the-box“. Na obrázku 14 je vidět vztah mezi Springem, Spring Bootem a uživatelem. První commit do veřejného repozitáře Spring Bootu na GitHubu provedl Phillip Webb v roce 2012. Následující rok se začínají objevovat další commity a v srpnu 2013 je publikována zpráva o dosažení prvního milníku, a tím je vydání první verze Spring Bootu. (WEBB, 2013)



Obrázek 14: *Umístění Spring Bootu v rámci frameworku Spring*[23]

Základní způsob, jak používat Spring Boot, je získání souboru „spring-boot-*.jar“ a jeho přidání na „ClassPath“[24]. To je však značně kontraproduktivní, a proto je Spring Boot plně kompatibilní s nástroji Maven a Gradle, které vše vyřeší v rámci závislostí. Základem je také embedovaný aplikační server Tomcat (případně Jetty nebo Undertow), takže není nutné produkovat WAR soubory (ale je to možné) a ty nasazovat do separátního aplikačního serveru, ale stačí pouze vyprodukovat a spustit JAR soubor.

[23] Zdroj: (WEBB, 2013).
[24] ClassPath je cesta ke složce nebo složkám, která je využívána ClassLoaderem při vyhledávání tříd v Java programu. (JAVAREVISITED, 2017)

Díky webové aplikaci start.spring.io je možné si nechat vygenerovat základní strukturu aplikace. Generovat je možné Maven nebo Gradle projekt. Také je možné si vybrat programovací jazyk, přičemž lze vybrat Javu, Kotlin nebo Groovy. Dále také verzi Spring Bootu, která bude základem aplikace. Autor této práce v době jejího psaní využívá verzi 1.5.12.

4.2.2 KOTLIN

Autor se rozhodl pro využití jazyka Kotlin z toho důvodu, že se mu líbí a již ho na několika školních projektech vyzkoušel. Zároveň autor pracuje v mezinárodní firmě, která v Kotlinu vyvíjí aplikaci, jež je nasazována do produkce po celém světě. Jedná se o staticky typovaný programovací jazyk pro psaní moderních multiplatformních aplikací, jež pro svůj běh využívá JVM. Kotlin lze využít pro vývoj klasických serverově orientovaných aplikací, od roku 2017 je to také oficiální jazyk pro Android aplikace. Zdrojový kód lze také transpilovat[25] do JavaScriptu a v neposlední řadě lze také využít technologii Kotlin/Native, díky níž se provádí kompilace přímo do nativních binárních souborů, které pro svůj běh nepotřebují JVM. I když lze využít jazyk i pro psaní front-endu, rozhodl se autor pro jeho využití výhradně na straně back-endu. Ve zdrojovém kódu 4 je uveden příklad jednoduchého zápisu programu v Kotlinu, který vypíše do konzole „Hello World!“. (KOTLIN, 2018a)

Zdrojový kód 4: *Hello world! zápis v Kotlinu*

```
package hello // optional package header

fun main(args: Array<String>) { // package-level function
    println("Hello World!") // semicolons are optional
}
```

Kotlin je vyvíjen jako open source firmou JetBrains, jež jej poprvé veřejnosti představila v roce 2010. První oficiální verze 1.0 byla vydána o šest let později v únoru 2016. Ke dni zpracování této práce je jazyk dostupný ve verzi 1.2.31, která je také použita při vývoji webové aplikace Zoumi.cz. (KOTLIN, 2018b)

[25] Proces, při kterém dochází ke čtení kódu v jednom jazyce a jeho ekvivalentní reprodukci v jiném jazyce. (SENGSTACKE, 2016)

Bezpečnost

Mezi některé z bezpečnostních prvků patří takzvaně „null-safety“, což znamená, že se snaží eliminovat výjimku typu „NullPointerException“ (dále NPE) už během kompilace. Jediné možné zdroje pro NPE jsou:

- explicitní volání „throw NullPointerException()“,
- využití operátoru !!, jenž převede každou hodnotu na nenulový typ, případně vyvolá výjimku NPE pokud je hodnota null,
- nekonzistentní data, například přístup k neinicializované proměnné během inicializace objektu,
- při operacích využívajících kód z Javy.

4.2.3 THYMELEAF

O samotné vykreslení HTML stránek se stará framework Thymeleaf. Jedná se o moderní serverový Java XML/XHTML/HTML5 šablonovací nástroj. Používání je založeno na využívání XML tagů a atributů. Thymeleaf je skvěle integrován s výše zmíněným Spring frameworkem. (THYMELEAF, 2017) V rámci projektu stačí pouze definovat závislost na Thymeleaf a začít ho používat. Přidání závislosti v souboru pom.xml (Maven) je vidět ve zdrojovém kódu 5.

Zdrojový kód 5: *Maven závislost na Thymeleaf pro Spring Boot*

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

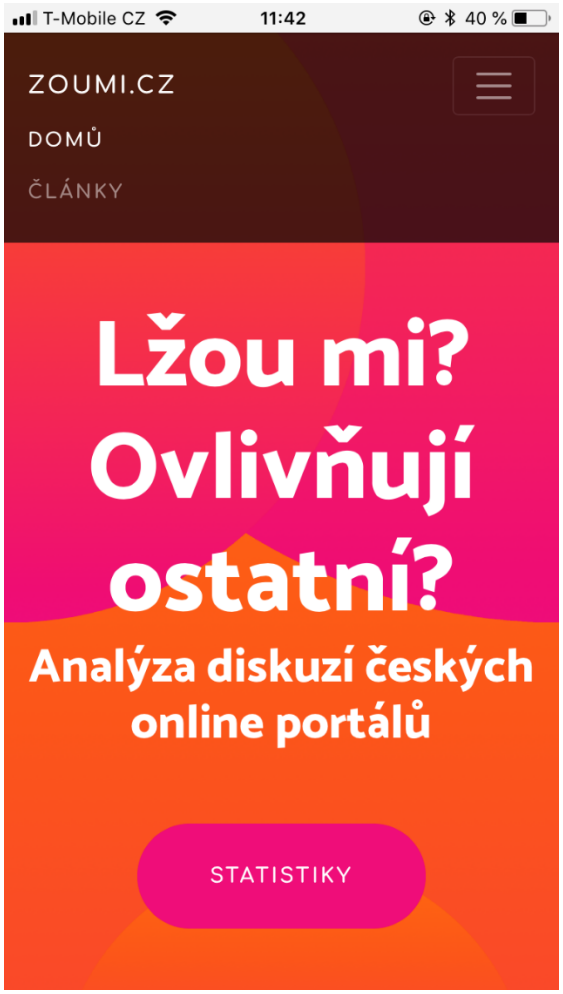
4.2.4 POSTGRESQL

Veškerá data v rámci aplikace jsou uložena v databázi PostgreSQL. Jedná se o open source objektově-relační databázový systém. Je vyvíjen „PostgreSQL Global Development Group“, což je skupina různých společností a individuálních přispěvatelů. (POSTGRESQL, 2018)

4.2.5 BOOTSTRAP

Jedná se o webový front-end framework, jež je používán pro vytváření webových stránek a webových aplikací. Kombinuje HTML, CSS a Javascript, díky čemuž lze jednoduše vytvářet příjemné responzivní uživatelské rozhraní. V rámci aplikace je využit Bootstrap 4 s využitím

volně dostupných šablon „One Page Wonder“ a „Blog Post“. Aplikace je díky využitému frameworku responzivní a přizpůsobí se i mobilnímu zařízení, jak je vidět na obrázku 15.



Obrázek 15: *Zoumi.cz na mobilním zařízení*

4.2.6 APACHE MAVEN

Jedná se o nástroj pro kompletní správu projektu, to zahrnuje například řízení a automatizaci sestavení aplikace. Maven je využíván hlavně pro Java aplikace a kromě toho, že ho lze využít v rámci příkazové řádky, je také integrován do řady IDE. K popsání projektu slouží XML soubor pom.xml, který obsahuje popis projektu, závislosti projektu na knihovny a další moduly, také popis sestavení a potřebné rozšíření. (MAVEN, 2018) V současné době je oblíbenou alternativou k Mavenu nástroj Gradle. V rámci této práce je použit Maven. To tedy znamená, že ho lze využít i pro kompilaci zdrojového kódu v Kotlinu, pouze je nutné se řídit doporučeným postupem z oficiálního webu[26]. V případě této práce bylo nutné provést kompilaci zdrojových kódů v Kotlinu a zároveň v Groovy (testy). Autor čelil problému, kdy

[26] Zdroj: (KOTLIN, 2018c).

ihned po zkompilování Groovy došlo k jejich vymazání a výstup obsahoval pouze zkompilovaný Kotlin kód. Jak nakonec z dokumentace vyplynulo, v případě kombinace více jazyků je nutné spustil kompilaci Kotlinu vždy jako první.

4.3 POPIS ARCHITEKTURY APLIKACE

Následující kapitoly obsahují popis toho, jak fungují kritické části aplikace. Aplikace je robustní a není zde popsáno vše, avšak kód je veřejně dostupný, takže si ho čtenář může volně procházet. Také je nastíněno, jak vypadá produkční prostředí spuštěné aplikace. Při vývoji aplikace se autor snažil aplikovat pravidla čistého kódu tak, jak je popsal Martin. (MARTIN, 2009)

4.3.1 STAHOVÁNÍ DAT Z EXTERNÍ DATABÁZE

Portály, články a jejich komentáře se stahují z externí databáze. Jejich získávání je součástí diplomové práce jiného studenta. Aplikace Zoumi.cz pouze jednou denně zjistí, jestli v externí databázi nejsou nová data, a ta případně stáhne. I když jsou modely téměř shodné, rozhodl se autor pro stahování do své vlastní databáze, aby nebyl závislý na funkčnosti externí databáze.

Každou noc v 1.00 se začínají aktualizovat data. Práce s externí databází je obsažena ve třídě DataDownloaderService. Postup získávání dat je následující:

1. zjisti počet portálů v externí databázi a porovnej s počtem v lokální databázi,
    a. pokud jsou stejné, pokračuj na krok 2,
    b. pokud externí databáze obsahuje více portálů, zjisti podle ID, jaké má navíc a ty vytvoř lokálně. Pokračuj na krok 2.
2. zjisti počet článků v externí databázi a porovnej s počtem v lokální databázi,
    a. pokud jsou stejné, pokračuj na krok 3,
    b. zjisti datum vytvoření nejnovějšího článku v lokální databázi,
    c. z externí databáze stáhni všechny články, které mají datum vytvoření vyšší než datum z bodu b,
    d. pro každý nový článek stáhni všechny jeho komentáře.
3. data jsou aktuální, stahování končí.

Algoritmus byl několikrát upravován kvůli problémům s pamětí a nízkou rychlostí při velkém počtu komentářů. Problém byl v tom, že Hibernate si ukládá do vyrovnávací paměti nově

vytvořené instance. To lze vyřešit dávkovým zpracováním. V této práci je nastaveno, že po každém třicátém komentáři dojde k nahrání do databáze a vyčištění instance Session[27].

4.3.2 ALGORITMUS PRO VÝPOČET PODOBNOSTI

Toto je nejkritičtější část aplikace. Každý den ve 3.00 dojde ke spuštění výpočtu podobnosti nad sadou nově získaných komentářů. Pro výpočet podobnosti využívá autor externí knihovnu java-string-similarity[28]. Tato knihovna obsahuje několik algoritmů pro výpočet podobnosti a vzdálenosti mezi texty. V rámci této práce je používán Jaccardův koeficient podobnosti, který na stupnici od 0 do 1 vyjadřuje podobnost mezi dvěma texty (kdy 1 znamená stejný text). Prahová hodnota, od které jsou komentáře označeny jako podobné, je nastavena na 0,6. Komentář, který je označen jako podobný, je pak uložen do databáze jako instance modelu SimilarComment, nesoucí si informace o identifikátorech daného komentáře a rodičovského článku. Porovnávají se vždy všechny komentáře mezi sebou, jež patří jednomu článku. Pokud má tedy článek 1000 komentářů, je nutné provést 1000×1000 operací což je 1 000 000 operací porovnání. Proto je naimplementováno porovnávání ve více vláknech, aby byl výpočet rychlejší. Hlavní implementace je uvedena v příloze A této práce. Jak je v konfiguraci vidět, vždy je alokován počet vláken rovnající se počtu dostupných procesorů.

4.3.3 POPIS CONTROLLERU APLIKACE

Aplikace disponuje dvěma controllery. Hlavní, třída AppController, slouží k odbavování požadavků na zobrazování různých stránek. Další, třída ApiController, je pouze pro testovací účely manuálního spouštění stahování nových dat, výpočtu podobnosti a přepočítání počtu komentářů. Nejzajímavější a zároveň nejdelší je metoda, která odbavuje požadavky na výpis všech komentářů a jejich filtrování. Tato metoda je přiložena v rámci přílohy B. Jak je vidět, obsahuje několik vstupních parametrů. Jedná se o GET požadavek, takže parametry jsou předávány v rámci URL. V tuto chvíli je to vhodné, protože se nepřenáší citlivé údaje a zároveň je možné uložit si takovou URL pro budoucí použití. Dále je vidět vyřešení řazení, vyhledávání podle klíčových slov, podle portálů a také vyřešení stránkování pomocí pomocné třídy Pager.

[27] Zdroj: (HIBERNATE, 2004).
[28] Zdroj: (GITHUB, 2018b).

4.3.4 POPIS POHLEDŮ APLIKACE

Aplikace disponuje třemi pohledy:

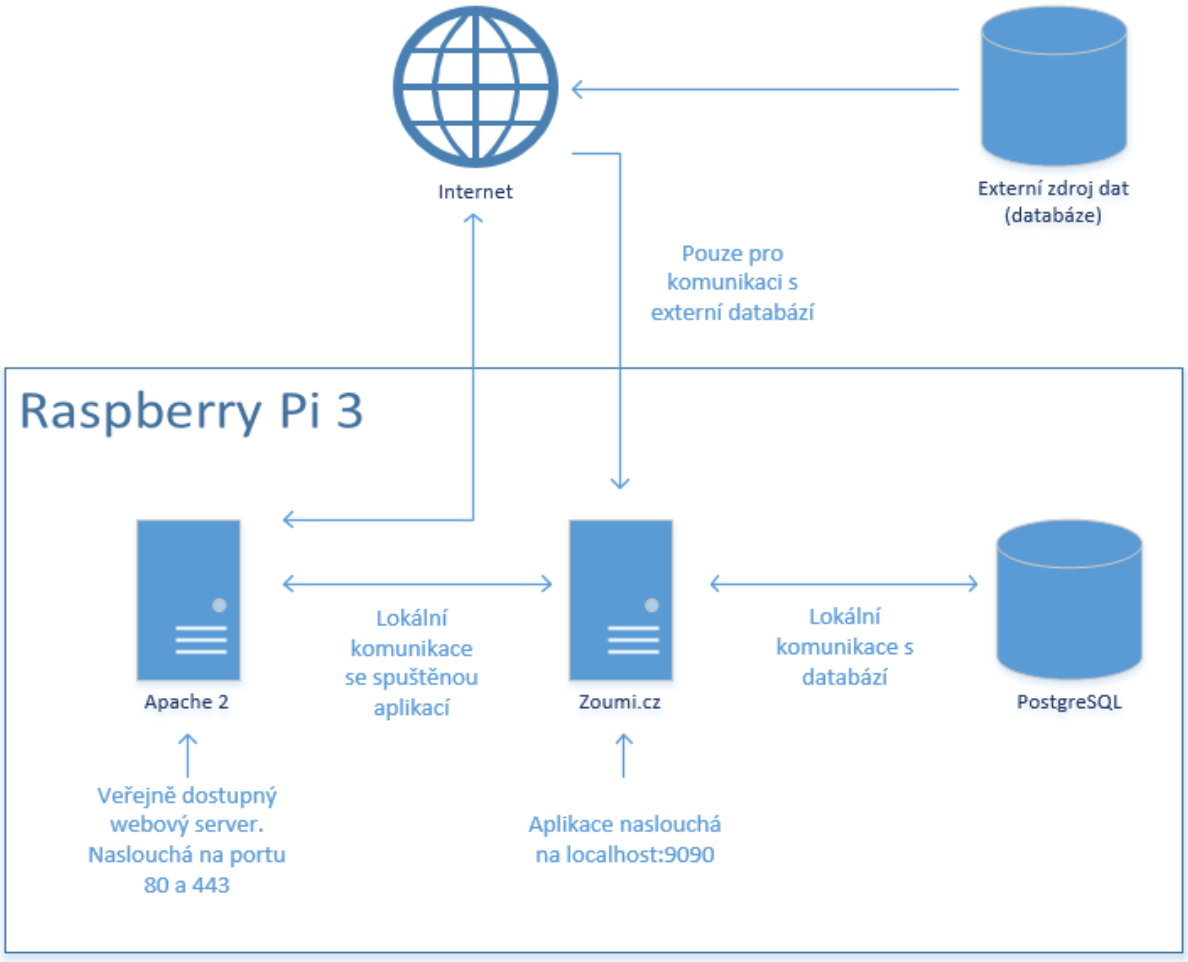
- index.html,
- article_list.html,
- article.html.

Na obrázku 15 je vidět náhled na index.html. Na tomto pohledu se dynamicky mění počet všech komentářů a také dva ukazatele o počtu podobných komentářů. Další pohled article_list.html je vidět na obrázku 13 a obsahuje výpis článků s možností různého filtrování a vyhledávání. Poslední, article.html, slouží k zobrazení konkrétního článku. Součástí náhledu je titulek, odkaz na originální zdroj a anotace. Dále je zde informace o celkovém počtu komentářů a celkovém počtu komentářů podobných. Z právních důvodů nejsou tyto komentáře veřejně dostupné. Jak tento pohled vypadá je vidět na obrázku 16.

Obrázek 16: *Detail článku v aplikaci Zoumi.cz*

4.3.5 ARCHITEKTURA PRODUKČNÍHO PROSTŘEDÍ APLIKACE

Jak již bylo řečeno, autor si aplikaci sám hostuje na svém serveru. Byť by nic nebránilo pouze spustit samotné .jar, které by dokázalo obsluhovat http požadavky také, rozhodl se autor pro řešení, kdy mezi požadavky a aplikaci vstupuje ještě webový server Apache. Ten pomocí správného nastavení proxy kontaktuje běžící aplikaci. Nástin architektury je vidět na obrázku 17. Samotná aplikace naslouchá na portu 9090. Konfigurační soubor pro stránku www.zoumi.cz a webový server Apache je uveden v příloze C této práce. Databáze je také nainstalována lokálně a není tak nutné jí vystavovat do internetu.



Obrázek 17: *Architektura produkčního prostředí*

Přístup k aplikaci je nastaven tak, aby vždy vyžadoval zabezpečené spojení pomocí HTTPS. Pokud se uživatel pokusí přistoupit na nezabezpečenou komunikaci, je přesměrován vždy na HTTPS. Server disponuje i platným certifikátem, díky kterému není uživatel vystaven upozornění prohlížeče. Vše se mu správně načte a v závislosti na prohlížeči je také náležitě informován o zabezpečené komunikaci. Konkrétně pro Google Chrome je to vidět na obrázku 18.

Obrázek 18: *Adresní řádek informující o zabezpečeném připojení*

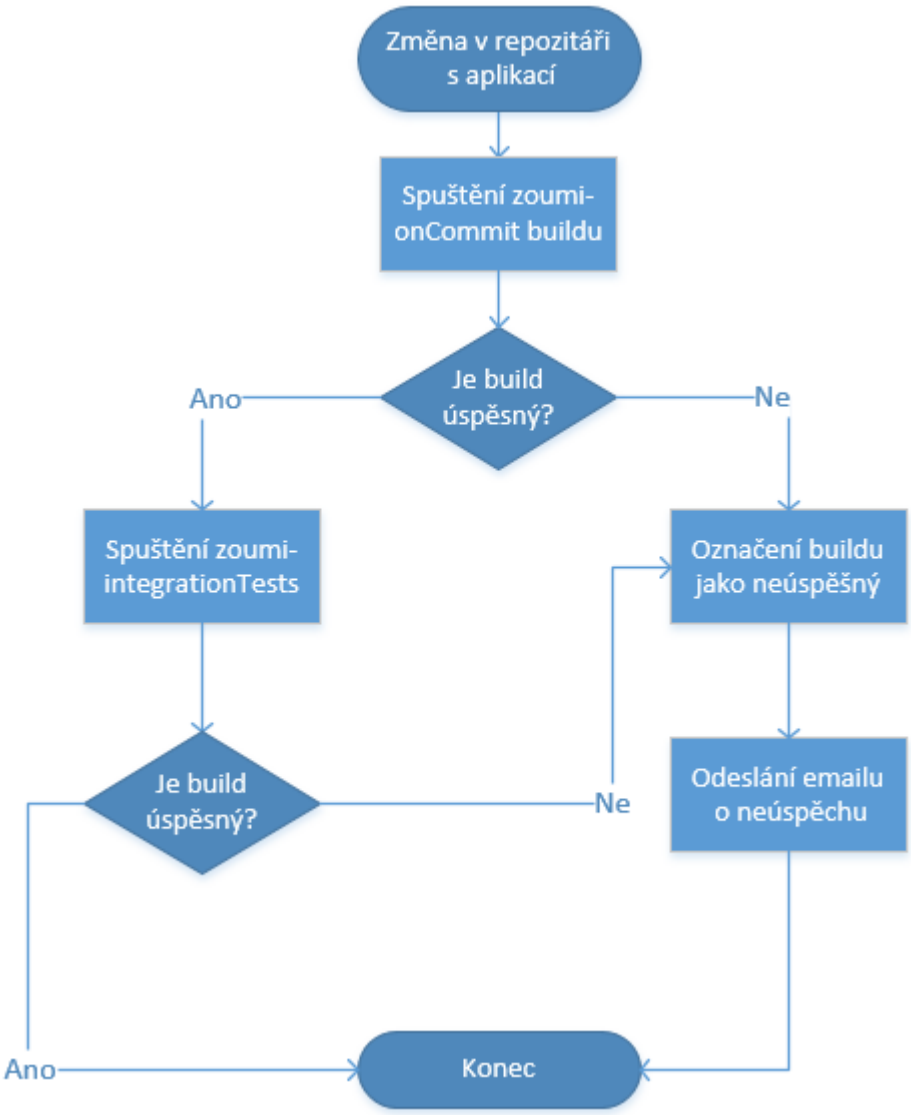
Certifikát je získán od certifikační autority Let's Encrypt, díky níž může mít každý svůj vlastní certifikát zdarma.

4.4 PROCES KONTINUÁLNÍ INTEGRACE APLIKACE

Jako hlavní server pro kontinuální integraci je využíván Jenkins. Ten si autor také sám hostuje a spravuje. Webové rozhraní přímo této konkrétní instance je na obrázku 7 a zahrnuje i přehled jobů, které jsou definovány. Pro samotnou kontinuální integraci jsou nejdůležitější joby:

- zoumi-onCommit,
- zoumi-integrationTests.

Proces kontinuální integrace aplikace Zoumi.cz je popsán vývojovým diagramem na obrázku 19.



Obrázek 19: *Vývojový diagram CI aplikace Zoumi.cz*

Jenkins je nainstalovaný jako systémový balíček a popis instalace je v kapitole 4.5.1. Architektura je nastavena podobně jako u samotné aplikace, to znamená, že Jenkins je spuštěn pouze lokálně a o vyřizování požadavků se stará webový server Apache. Konfigurační soubor pro stránku ci.zoumi.cz/jenkins a webový server Apache je uveden v příloze D.

Webové rozhraní pro Jenkins je dostupné na adrese ci.zoumi.cz/jenkins. I tato aplikace komunikuje pouze šifrovaně a má nastavený platný certifikát od certifikační autority Let's Encrypt.

4.4.1 ZOUMI-ONCOMMIT

Nové sestavení v rámci tohoto jobu je spuštěno vždy, pokud je přidána nová změna do repozitáře na GitHubu. Job je nastaven tak, že čeká na zavolání „web hooku" přímo

z GitHubu. V takovou chvíli se spustí nové sestavení aplikace, jež zahrnuje i vykonání jednotkových testů. Není tak nutné periodicky zjišťovat případné změny v kódu. Konfigurace tohoto jobu je uvedena v příloze E.

4.4.2 ZOUMI-INTEGRATIONTESTS

Jedná se o job, který definuje spouštění sestavení s integračními testy. Sestavení v rámci tohoto jobu je spuštěno jen tehdy, pokud je poslední sestavení v rámci zoumi-onCommit dokončeno úspěšně. Konfigurace tohoto jobu je uvedena v příloze F. Aby Maven vynechal jednotkové testy, je nutné uvést přepínač „-DskipUTs=true“.

4.4.3 NASAZENÍ NOVÉ VERZE

Nasazení nové verze aplikace do produkčního prostředí je automatizováno, ale je nutné ho vyvolat manuálně kliknutím na tlačítko, respektive na dvě. Autor tedy praktikuje přístup „Continuous Delivery“, který byl popsán v rámci kapitoly 3.4.1. Používá k tomu job zoumi-deployFreestyle. Typ jobu je „Freestyle“ a vykonává následující:

1. z aktuálních zdrojových kódů provede sestavení souboru .jar,
2. tento soubor zkopíruje do /var/www/zoumi-production,
3. v té samé složce spustí start.sh, jež obsahuje startovací skript,
4. aplikace je nyní spuštěna.

V této části se autor potýkal s problémem, kdy při každém vytvoření nového procesu uživatelem jenkins byl takový proces ihned po skončení sestavení na Jenkinsu ukončen. Oficiálním řešením je přidání proměnné „JENKINS_SERVER_COOKIE“ s hodnotou „dontkill“[29].

Konfigurace jobu zoumi-deployFreestyle je uvedena v příloze G.

4.4.4 POMOCNÉ JOBY

Poslední dva zbývající joby nesou název zoumi-startProduction a zoumi-stopProduction. Jedná se o spuštění startovacího skriptu start.sh, respektive zaslání POST requestu na koncový bod „/shutdown“. Během vývoje se stávalo, že aplikaci bylo nutné restartovat, proto si autor tento proces zautomatizoval. Pro zastavení běžící aplikace se volá:

```
curl -X POST http://localhost:9090/shutdown
```

[29] Zdroj: (STACKEXCHANGE, 2018).

Start aplikace probíhá téměř shodně jako nasazení nové verze, pouze se vynechá první a druhý bod.

4.5 REPRODUKCE NASAZENÍ

Následující kapitoly obsahují popis, jak provést nasazení jak CI prostředí, tak samotné aplikace.

4.5.1 JENKINS CI

Příprava prostředí

Díky využitému programovacímu jazyku Kotlin a výběru CI nástroje Jenkins je možné využít pro nasazení téměř jakoukoliv dostupnou platformu. V této práci proběhlo nasazení CI serveru Jenkins do prostředí systému Linux, konkrétně Raspbian Lite 4.9.59-v7. Do systému je nezbytně nutné nainstalovat Java JDK a nastavit proměnnou JAVA_HOME (pokud se tak nestane automaticky) pomocí následujících příkazů:

```
sudo apt-get install openjdk-8-jdk
export JAVA_HOME=/usr/lib/jvm/<aktuální verze Javy>
export PATH=$PATH:$JAVA_HOME/bin
```

Dalším krokem je instalace nástroje Apache Maven (popsáno v kapitole 4.2.6), což je provedeno jednoduchým příkazem:

```
apt-get install maven
```

V této chvíli jsou splněny minimální předpoklady pro nainstalování samotného CI serveru Jenkins, instalace je provedena následujícími příkazy, kompletní popis je pak uveden v oficiální dokumentaci: (KAWAGUCHI, 2009)

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key |
sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/
> /etc/apt/sources.list.d/jenkins.list'
sudo apt-get update
sudo apt-get install jenkins
```

V této chvíli je Jenkins dostupný na adrese http://localhost:8080/jenkins, kde jsou také informace, jak dále pokračovat s konfigurací.

Zdrojové kódy aplikace jsou umístěny ve službě GitHub, která, jak již z názvu plyne, využívá verzovací systém git. Je tedy nutné nainstalovat i tento software pomocí příkazu:

```
apt-get install git
```

Konfigurace Jenkins

Jenkins je nyní připraven pro nastavení prvního jobu. Konkrétně jsou joby použité v praktické části popsány v rámci kapitol 4.4.1, 4.4.2 a 4.4.3.

Ve webovém rozhraní Jenkinsu je nutné kliknout na „New Item“. Na nově otevřené stránce vyplnit jméno jobu a vybrat typ projektu „Pipeline“. V tuto chvíli je vytvořen nový job, který je však prázdný a je potřeba provést jeho konfiguraci. V části „General“ stačí vyplnit „Project url“ v konfiguraci „GitHub project“ na https://github.com/f0lik/zoumi/. Dále se musí nastavit, za jakých okolností se spustí build. Autor využívá přímo web hook z GitHubu, ale vzhledem k svázání pouze s aktuální instancí zde doporučuje zvolit „Poll SCM“. Díky tomu bude sám Jenkins ve stanoveném čase (zadaném pomocí cronu) zjišťovat, jestli v repozitáři s kódy aplikace nenastala nějaká změna. Poslední část je definice „Pipeline script“, ten je nutné vyplnit na základě vzoru z přílohy E této práce.

4.5.2 APLIKACE ZOUMI.CZ

Aplikaci lze spustit několika způsoby a také je možné jí spustit pouze lokálně, nemusí být nutně dostupná v rámci Internetu. Prostředí, na kterém bude aplikace spuštěna, musí obsahovat instalaci Javy, a to ve verzi 8. Aplikace pro svůj běh také vyžaduje spuštěnou a dostupnou databázi PostgreSQL na standardním portu 5432. Autor zaručuje funkčnost pro verzi 9.6.7, se kterou je aplikace vyvíjena. Před spuštěním samotné aplikace je nutné vytvořit databázi s názvem „zoumidb“ a provést import zálohy databáze, ten je obsahem přílohy na CD v souboru zoumidb_ddl.sql.

Zabalená aplikace z přílohy na CD

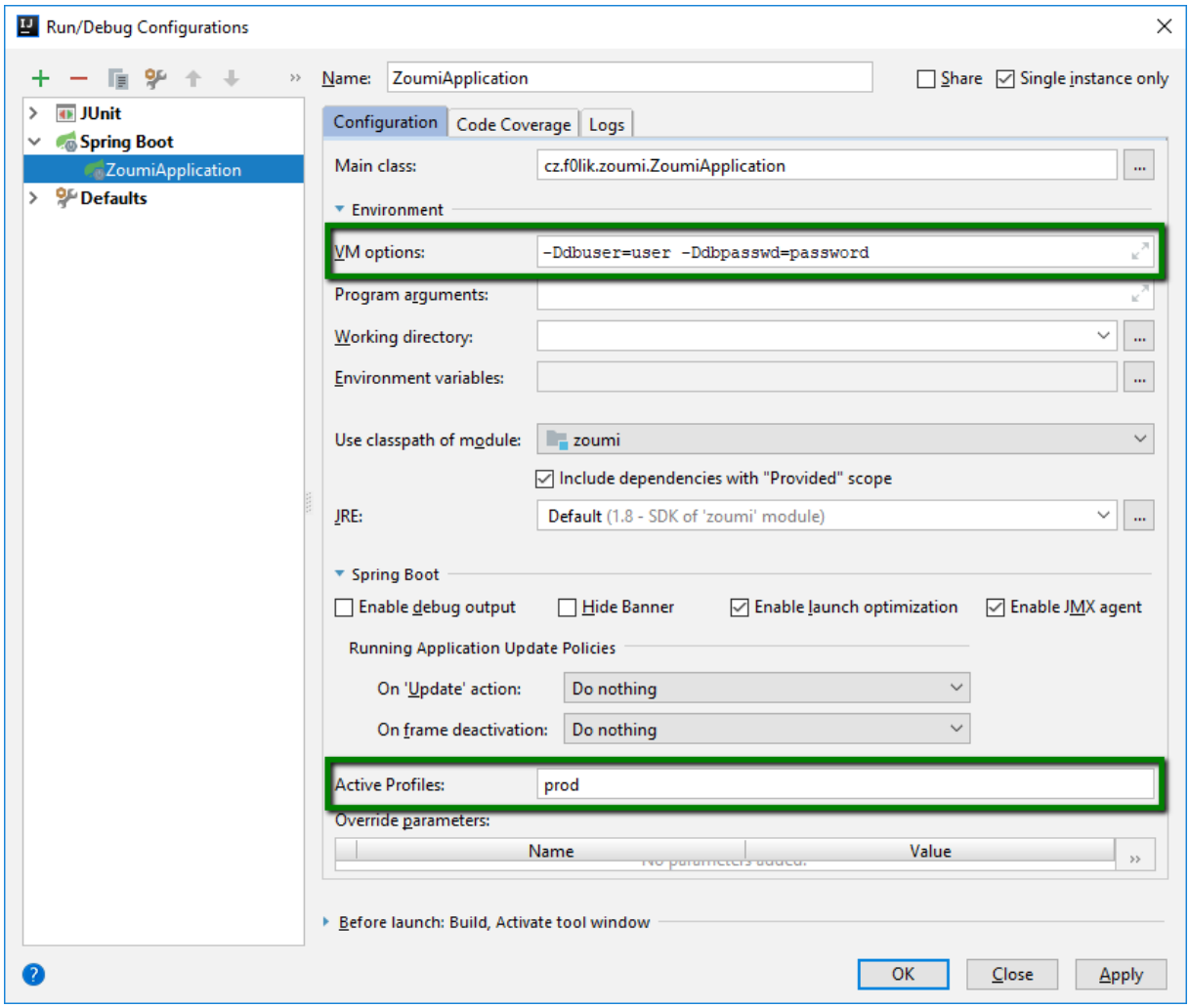
V rámci příloh dodaných na CD je kromě zdrojových kódů obsažen také balíček zoumi-1.0.0-SNAPSHOT.jar. Pomocí tohoto souboru lze aplikaci spustit s minimální konfigurací pomocí následujícího příkazu:

```
java
-Ddbuser=user
-Ddbpasswd=password
-Dspring.profiles.active=prod
-jar zoumi-1.0.0-SNAPSHOT.jar
```

Pokud je správně definován uživatel a heslo k databázi, proběhne start aplikace, která je po nějaké chvíli dostupná na adrese http://localhost:8090/.

Ze zdrojových kódů

V závislosti na IDE je možné aplikaci spustit také přímo ze zdrojových kódů. Pomocí IDE je nutné otevřít projekt, který je přiložen na CD, nejčastěji pomocí souboru pom.xml. Samotné IDE již provede konfiguraci projektu, případně je nutné nalézt postup, jak v patřičném IDE spustit Spring Boot aplikaci. V každém případě je nutné definovat databázového uživatele a heslo, dále také zvolit jako aktivní profil „prod“. Konfigurace pro IntelliJ IDEA je uvedena na obrázku 20. Důležité části, které je nutné vyplnit manuálně, jsou zvýrazněny pomocí zeleného obdélníku.

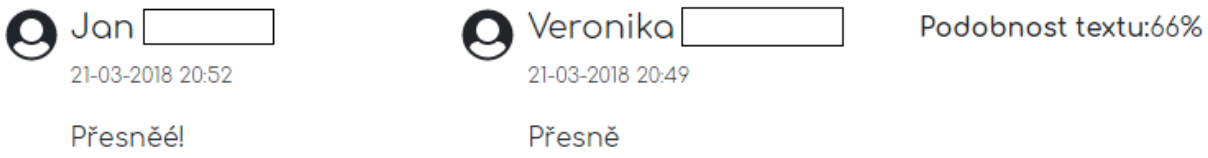


Obrázek 20: *Nastavení projektu v IntelliJ IDEA*

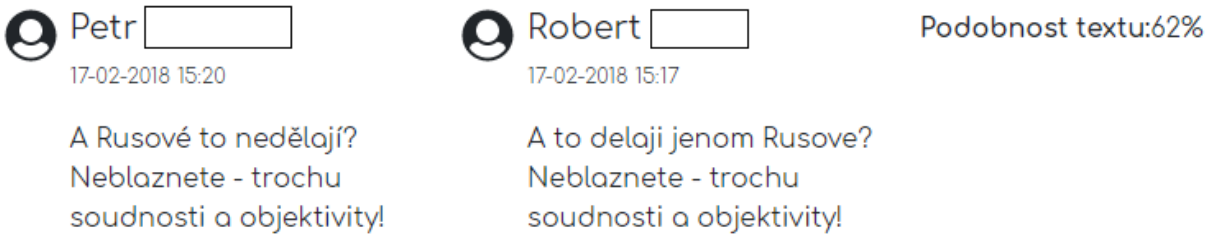
4.6 PREZENTACE VÝSLEDKŮ

Aplikace ke dni 25. 4. 2018 disponuje databází obsahující 581 747 komentářů, které jsou získány ze třech českých zpravodajských portálů. Z celkového počtu je 27 902 označeno jako podobných. Jsou to takové komentáře, u kterých vyšel index podobnosti větší než 60 %. Vzhledem k surovosti dat je však toto číslo výrazně ovlivněno komentáři z jednoho konkrétního portálu, ve kterém neexistují u článku diskuzní vlákna. Lidé zde tak odpovídají ostatním diskutujícím způsobem, že zkopírují celý komentář, na který reagují, a ten vloží do své odpovědi. Pokud je tento portál ze statistik vynechán, celkový počet komentářů je 148 886, z čehož je 1 137 označeno jako podobných. Tato práce již dále komentáře neanalyzuje a nezpracovává. Následují ale ukázky některých komentářů, jež byly označeny jako podobné.

Na obrázku 21 je vidět označený komentář s podobností textu 66 %. Na takovém komentáři není nic zajímavého, ale díky stejnému slovu byl označen jako podobný. Naopak obrázek 22 může vést k otázce, proč jiný autor pouze pozměnil první větu a takový komentář zveřejnil?



Obrázek 21: *Podobný komentář 1*



Obrázek 22: *Podobný komentář 2*

ZÁVĚR

Pokud vývojáři v týmu pracují na stejném produktu, musí všichni respektovat určitá pravidla. S příchodem metodiky kontinuální integrace jsou tato pravidla ještě striktnější, jak bylo popsáno v teoretické části. Velice důležité je správné využívání verzovacího systému, který je nutný pro nasazení CI. Součástí vývoje softwaru musí být také testy, které byly popsány i s uvedením správného přístupu, jež je označován jako testovací pyramida, a zároveň špatného přístupu označovaného jako zmrzlinový kornout.

Nastavení firemních procesů, ale i samotných nástrojů, se může zdát ze začátku jako zdlouhavé a náročné, na druhou stranu přínos je dlouhodobý a otevírá dveře k nasazení dalších metodik, jako jsou Continuous Delivery (také využit v rámci praktické části) a Continous Deployment. Díky existenci mnoha nástrojů pro nasazení CI však i tento krok může být velice jednoduchý. V rámci této práce proběhlo porovnání mezi nástroji Jenkins, Travis CI a Cirecle CI, přičemž poslední dvě jsou online služby dostupné zdarma. Jak autor vyzkoušel, nasazení vyvíjené aplikace do CI procesu v těchto online službách zabralo pouze několik jednotek minut.

V rámci praktické části byla vyvinuta aplikace Zoumi.cz pro analýzu podobnosti mezi texty z komentářů na českých zpravodajských portálech. Použité technologie jsou velice moderní a autor vyzdvihuje hlavně použití programovacího jazyka Kotlin, jež se stává čím dál více populární. Aplikace využívá extérní knihovnu implementující algoritmus pro výpočet Jaccardova koeficientu podobnosti. Během celého vývoje byla aplikace kontinuálně integrována v rámci serveru Jenkins CI. Ten si autor sám spravuje a celý proces kontinuální integrace si navrhl a implementoval sám. Jeho popis i s odkazy na přílohy byl popsán v rámci praktické části a je tak možné ho reprodukovat. Celý projekt je kombinací různých technologií, ale i programovacích jazyků. Například integrační testy jsou psané v jazyce Groovy a využívají framework Spock. To vedlo k problémům během sestavování aplikace, které autor vyřešil a popsal v rámci této práce. Autor práce se rozhodl aplikaci veřejně publikovat, je tak dostupná všem na Zoumi.cz. Mezi uživatelem a serverem je vynuceno navázání zabezpečeného připojení, což by měl být v dnešní době standard, a to i díky dostupnosti certifikátů, jež je možné získat zdarma od certifikační autority Let’s Encrypt. Smutný je fakt, že díky právním záležitostem není možné na webu publikovat žádné texty komentářů. Je tak pouze uveden údaj, zdali jsou u daného článku některé komentáře podobné, a pokud ano, tak i jejich počet.

POUŽITÁ LITERATURA

1. AGILE MANIFESTO. 2001. *Twelve Principles of Agile Software*. Únor 2001 [online]. [cit. 2018-03-29]. Dostupné z: http://agilemanifesto.org/iso/en/manifesto.html
2. ALISTER, Scott. 2018. *Testing Pyramids & Ice-Cream Cones*. WatirMelon blog. [online]. [cit. 2018-03-28]. Dostupné z: https://watirmelon.blog/testing-pyramids/.
3. AMAZON. 2018. *Cloud Computing with Amazon Web Services*. [online]. [cit. 2018-03-28]. Dostupné z: https://aws.amazon.com/what-is-aws/.
4. BECK, Kent. 1999. Embracing Change With Extreme Programming. *Computer*. Volume 32, Issue 10. 70-77 s. ISSN: 0018-9162.
5. BECK, Kent. 2002. *Test Driven Development: By Example*. První vydání. Addison-Wesley Professional. 240 s. ISBN: 978-0321146533.
6. BOOCH, Grady. 1994. *Object-Oriented Analysis and Design with Applications*. 2. vydání. Redwood City: Benjamin/Cummings Publishing Company. 589 s. ISBN 0-8053-5340-2.
7. CIRCLECI. 2018. *2.0 Docs*. [online]. [cit. 2018-03-28]. Dostupné z: https://circleci.com/docs/2.0/
8. DUVALL, Paul M., MATYAS, Steve a Andrew GLOVER. 2007. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley. 283 s. ISBN: 032133638.
9. FOWLER, Martin. 2000. *Continuous Integration (original version)*. Martin Fowler – blog. 10.9.2000 [online]. [cit. 2018-03-27]. Dostupné z: https://www.martinfowler.com/articles/originalContinuousIntegration.html
10. FOWLER, Martin. 2006. *Continuous Integration*. Martin Fowler – blog. 1.5.2006 [online]. [cit. 2018-03-27]. Dostupné z: https://martinfowler.com/articles/continuousIntegration.html
11. GEBISH. 2018. *What is it and What does it look like?* [online]. [cit. 2018-03-28]. Dostupné z: http://www.gebish.org/.
12. GITGEAR. 2018. *Jenkins Users! Rejoice!* [online]. [cit. 2018-03-28]. Dostupné z: http://gitgear.com/xfd/.
13. GITHUB. 2018a. *f0lik/zoumi*. [online]. [cit. 2018-03-28]. Dostupné z: https://github.com/f0lik/zoumi
14. GITHUB. 2018b. tdebatty/java-string-similarity [online]. [cit. 2018-03-28]. Dostupné z: https://github.com/tdebatty/java-string-similarity.
15. MYERS, Glenford J.. 2004. *The Art of Software Testing*. 2. vydání. New Jersey: John Wiley & Sons, Inc. 234 s. ISBN 0-471-46912-2.

16. GUSFIELD, Dan. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press: New York. 534 s. ISBN:0-521-58519-8.
17. HIBERNATE. 2004. *Batch processing*. Community Documentation. [online]. [cit. 2018-03-28]. Dostupné z: https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/batch.html
18. HIGHSMITH, Jim a Alistair, COCKBURN. 2001. Agile Software Development: The Business of Innovation. *Computer*. Volume 34, Issue 9. 120-127 s. ISSN: 0018-9162.
19. HLAVA, Tomáš. 2011. *Fáze a úrovně provádění testů*. Testování softwaru. 21.8.2011 [online]. [cit. 2018-03-28]. Dostupné z: http://testovanisoftwaru.cz/category/metodika-testovani/druhy-typy-a-kategorie-testu/#acceptance.
20. HUJER, Martin. 2012. *Kontinuální integrace při vývoji webových aplikací v PHP*. Bakalářská práce. Vysoká škola ekonomická v Praze, fakulta informatiky a statistiky. 69 s. Vedoucí práce: Ing. Jan Mittner.
21. HUMBLE, Jez, NORTH, Dan a Chris READ. (2006). *The deployment production line*. Agile Conference, 23-28 července 2006. ISBN: 0-7695-2562-8.
22. IN FLOW. 2008. Troll Internetový. *In flow – information journal*. 20.5.2008 [online]. [cit. 2018-03-28]. Dostupné z: http://www.inflow.cz/troll-internetovy.
23. INTEROUTE. 2018. *What Is PaaS?* [online]. [cit. 2018-03-27]. Dostupné z: https://www.interoute.com/what-paas
24. ISTQB EXAM CERTIFICATION. 2013. *What is V-model- advantages, disadvantages and when to use it?* 2013 [online]. [cit. 2018-03-29]. Dostupné z: http://istqbexamcertification.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/
25. JAVAREVISITED. 2017. *How to Set Classpath for Java on Windows and Linux*. 28.1.2017 [online]. [cit. 2018-03-28]. Dostupné z: http://javarevisited.blogspot.cz/2011/01/how-classpath-work-in-java.html.
26. JENKINS. 2018. *Jenkins User Handbook*. [online]. [cit. 2018-03-29]. Dostupné z: https://jenkins.io/doc/book/getting-started/
27. KADLEC, Václav. 2004. *Agilní programování: metodiky efektivního vývoje softwaru*. Brno: Computer Press. 280 s. ISBN isbn:80-251-0342-0.
28. KAWAGUCHI, Kohsuke. 2007. *Hudson*. JavaOneSM Conference. [online]. [cit. 2018-03-28]. Dostupné z: https://web.archive.org/web/20140701020639/https://www.java.net//blog/kohsuke/archive/20070514/Hudson%20J1.pdf

29. KAWAGUCHI, Kohsuke. 2009. *Installing Jenkins on Ubuntu*. ATLASSIAN: wiki Jenkins. [online]. [cit. 2018-03-28]. Dostupné z: https://wiki.jenkins.io/display/JENKINS/Installing+Jenkins+on+Ubuntu
30. KOTLIN. 2018a. *What does it look like?* [online]. [cit. 2018-03-28]. Dostupné z: https://kotlinlang.org/
31. KOTLIN. 2018b. *Reference*. [online]. [cit. 2018-03-28]. Dostupné z: https://kotlinlang.org/docs/reference/
32. KOTLIN. 2018c. *Using Maven.* [online]. [cit. 2018-03-28]. Dostupné z: https://kotlinlang.org/docs/reference/using-maven.html
33. LAPLANTE, Phillip A., NEILL, Colin J. 2004. The Demise of the Waterfall Model Is Imminent. *Queue - Game Development*. Volume 1. Issue 10. 10-15 s. ISSN:1542-7730
34. MARTIN, Robert C. 2009. *Čistý kód: návrhové vzory, redaktorování, testování a další techniky agilního programování*. Brno: Computer Press. 424 s. ISBN 978-80-251-2285-3.
35. MAVEN. 2018. *What is Maven?* The Apache Software Foundation. [online]. [cit. 2018-03-28]. Dostupné z: http://maven.apache.org/what-is-maven.html.
36. NIEDERWIESER, Peter. *Introduction - The Spock Framework TeamVersion 1.1*. Spock framework. [online]. [cit. 2018-03-28]. Dostupné z: http://spockframework.org/spock/docs/1.1/introduction.html
37. ONE STOP TESTING. 2018. *Introduction to Software Testing*. [online]. [cit. 2018-03-28]. Dostupné z: http://www.onestoptesting.com/introduction/
38. ORIGINAL SOFTWARE. 2018. *Types of User Acceptance Testing*. [online]. [cit. 2018-03-28]. Dostupné z: https://www.origsoft.com/solutions/user-acceptance-testing/types-of-user-acceptance-testing/
39. PATTON, Ron. 2001. *Software Testing*. Indianapolis, Indiana: Sams Publishing. 389 s. ISBN 0-672-31983-7.
40. PIERRE CARBONNELLE. 2018. *Top IDE index*. Březen 2018 [online]. [cit. 2018-04-20]. Dostupné z: https://pypl.github.io/IDE.html.
41. PITTET, Sten. 2018. *Continuous integration vs. continuous delivery vs. continuous deployment*. [online]. [cit. 2018-03-28]. Dostupné z: https://www.atlassian.com/continuous-delivery/ci-vs-ci-vs-cd.
42. POSTGRESQL. 2018. *What Is Postgresql and Why Use Postgresql?* 1.3.2018 [online]. [cit. 2018-03-28]. Dostupné z: https://www.postgresql.org/about/.

43. SANDOVAL, Kristopher. 2016. *What is the Difference Between an API and an SDK?* Nordic APIs. [online]. [cit. 2018-03-28]. Dostupné z: https://nordicapis.com/what-is-the-difference-between-an-api-and-an-sdk/
44. SENGSTACKE, Peleke. 2016. *JavaScript Transpilers: What They Are & Why We Need Them.* Scotch.io. 25.4.2016 [online]. [cit. 2018-03-28]. Dostupné z: https://scotch.io/tutorials/javascript-transpilers-what-they-are-why-we-need-them.
45. SHARMA, Lakshay. 2016. *Smoke Testing*. TOOLSQA. 9.5.2016 [online]. [cit. 2018-03-28]. Dostupné z: http://toolsqa.com/software-testing/smoke-testing/.
46. SCHWABER, Ken a Jeff SUTHERLAND. 2017. *The Scrum Guide*. [online]. [cit. 2018-03-29]. Dostupné z: https://www.scrumguides.org/download.html.
47. SLACK. 2018. *What is Slack?* [online]. [cit. 2018-03-28]. Dostupné z: https://get.slack.help/hc/en-us/articles/115004071768-What-is-Slack-.
48. SMARTBEAR SOFTWARE. 2018a. *What is Code Review?* [online]. [cit. 2018-03-27]. Dostupné z: https://smartbear.com/learn/code-review/what-is-code-review/
49. SMARTBEAR SOFTWARE. 2018b. *What Is Integration Testing*. [online]. [cit. 2018-03-28]. Dostupné z: https://smartbear.com/learn/automated-testing/what-is-integration-testing/.
50. STACK OVERFLOW. 2018. *Developer Survey Results 2018*. [online]. [cit. 2018-03-28]. Dostupné z: https://insights.stackoverflow.com/survey/2018
51. STACKEXCHANGE. 2018. *Running a background process in Pipeline job*. [online]. [cit. 2018-03-28]. Dostupné z: https://devops.stackexchange.com/questions/1473/running-a-background-process-in-pipeline-job
52. THYMELEAF. 2017. *Tutorial: Using Thymeleaf*. 4.11.2017 [online]. [cit. 2018-03-28]. Dostupné z: https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html
53. TOM PRESTON-WERNER. 2018. *Semantic Versioning 2.0.0*. [online]. [cit. 2018-03-28]. Dostupné z: https://semver.org/
54. TRAVIS CI. 2018a. *Built for every team*. [online]. [cit. 2018-03-28]. Dostupné z: https://travis-ci.com/plans
55. TRAVIS CI. 2018b. *Languages*. [online]. [cit. 2018-03-28]. Dostupné z: https://docs.travis-ci.com/user/languages/
56. TURNER, Dawn M. 2016. *Digital Authentication - the basics*. Cryptomathic. 1.8.2016 [online]. [cit. 2018-03-28]. Dostupné z: https://www.cryptomathic.com/news-events/blog/digital-authentication-the-basics.

57. VENNERS, Bill. 2018. *The Java Virtual Machine*. Artima. [online]. [cit. 2018-03-27]. Dostupné z: https://www.artima.com/insidejvm/ed2/jvm.html
58. VIM. 2018. *Vim - the ubiquitous text editor*. Dostupné z: https://vim8.org/.
59. VOCKE, Ham. 2018. The Practical Test Pyramid. 26.2.2018 [online]. [cit. 2018-03-28]. Dostupné z: https://martinfowler.com/articles/practical-test-pyramid.html.
60. WEBB, Phill. 2013. *Spring Boot – Simplifying Spring for Everyone*. Spring blog. 6.8.2013 [online]. [cit. 2018-03-28]. Dostupné z: https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone/
61. WIKIMEDIA COMMONS. 2005a. *Waterfall model*. 25.11.2005 [online]. [cit. 2018-03-29]. Dostupné z: https://commons.wikimedia.org/wiki/File:Waterfall_model.png
62. WIKIPEDIA COMMONS. 2005b. *V-Model (software development).* 1.10.2005 [online]. [cit. 2018-03-29]. Dostupné z: https://en.wikipedia.org/wiki/V-Model_(software_development)
63. ZAMBORSKÝ, Matúš. 2012. *Kontinuálna integrácia*. Diplomová práce. Masarykova univerzita, fakulta informatiky. 78 s. Vedoucí diplomové práce RNDr. Petr Švenda, Ph.D.

SEZNAM PŘÍLOH

Příloha A – *Algoritmus výpočtu podobnosti mezi komentáři*

```
fun compareArticleComments(articleId: Long) {
    val newerComments =
commentRepository.getNewCommentDTO(articleId)

    val allComments =
commentRepository.getCommentDTOList(articleId)
    val newFixedThreadPool =
Executors.newFixedThreadPool(Runtime.getRuntime().availableProce
ssors())

    similarComments = similarCommentRepository.findAll()
    val processedPairIds = HashSet<String>()

    markCommentAsNotNew(articleId)

    newerComments.forEach { newComment ->
        val callableSimilarityTasks =
ArrayList<Callable<Double>>()
        allComments.forEach inner@{ comment ->
            val pairKey = Math.min(newComment.getCommentId(),
comment.getCommentId()).toString() +
                    "_" + Math.max(newComment.getCommentId(),
comment.getCommentId())
            if (processedPairIds.contains(pairKey)) {
                return@inner
            }
            if (newComment.getCommentId() ==
comment.getCommentId()) {
                return@inner
            }
            if (newComment.getCommentText() ==
comment.getCommentText()) {
                return@inner
            }
            val callableSimilarityTask = Callable<Double> {
                checkCommentSimilarity(newComment, comment)
            }
            callableSimilarityTasks.add(callableSimilarityTask)
            processedPairIds.add(pairKey)
        }
        newFixedThreadPool.invokeAll(callableSimilarityTasks)
    }
    processedPairIds.clear()
    newFixedThreadPool.shutdown()
}
```

Příloha B – *Metoda controlleru pro vykreslení pohledu se všemi články*

```
@GetMapping("/articles")
fun getArticles(@RequestParam("pageSize") pageSize: Optional<Int>,
                @RequestParam("page")page: Optional<Int>,
                @RequestParam("sortAttribute") sortAttribute:
Optional<String>,
                @RequestParam("sortDirection") sortDirection:
Optional<String>,
                @RequestParam("search") search: Optional<String>,
                @RequestParam("portal")  choosedPortal: Optional<Int>):
ModelAndView {
    val modelAndView = ModelAndView("article_list")
    val evaluatedPageSize = when {
        pageSize.orElse(INITIAL_PAGE_SIZE) > 20 -> INITIAL_PAGE_SIZE
        else -> pageSize.orElse(INITIAL_PAGE_SIZE)
    }
    val evaluatedPage = if (page.orElse(0) < 1) INITIAL_PAGE else page.get()
- 1
    val evaluatedSortAttribute = sortAttribute.orElse("similarCommentCount")
    val evaluatedSortDirection =
sortDirection.orElse(Sort.Direction.DESC.toString())
    val evaluatedPortal = choosedPortal.orElse(-1)

    val articles = when {
        search.isPresent ->
articleService.listAllByPage(search.get().toLowerCase(),
                PageRequest(evaluatedPage, evaluatedPageSize,
                Sort.Direction.fromString(evaluatedSortDirection),
evaluatedSortAttribute))
        choosedPortal.isPresent ->
articleService.listAllByPortal(choosedPortal.get().toLong(),
                PageRequest(evaluatedPage, evaluatedPageSize,
                Sort.Direction.fromString(evaluatedSortDirection),
evaluatedSortAttribute))
        else -> articleService.listAllByPage(PageRequest(evaluatedPage,
evaluatedPageSize,
                Sort.Direction.fromString(evaluatedSortDirection),
evaluatedSortAttribute))
    }
    val pager = Pager(articles.totalPages, articles.number, 5)

    val portalIdNameMap = portalService.getPortalIdNameMap()
    portalIdNameMap![-1] = "--Všechny--"
    modelAndView.addObject("articles", articles)
    modelAndView.addObject("portalMap", portalIdNameMap)
    modelAndView.addObject("selectedPortal", evaluatedPortal)
    modelAndView.addObject("selectedPageSize", evaluatedPageSize)
    modelAndView.addObject("pageSizes", PAGE_SIZES)
    modelAndView.addObject("sortAttributes", sortAttributesMap)
    modelAndView.addObject("selectedSortAttribute", evaluatedSortAttribute)
    modelAndView.addObject("sortDirections", sortDirectionMap)
    modelAndView.addObject("selectedSortDirection", evaluatedSortDirection)
    modelAndView.addObject("pager", pager)
    modelAndView.addObject("version", applicationVersion)
    return modelAndView
}
```

Příloha C – *Konfigurační soubor pro webovou stránku Zoumi.cz*

```
<VirtualHost *:80>
        ServerAdmin fol.pavel@gmail.com
        ServerName zoumi.cz
        ServerAlias www.zoumi.cz
        RewriteEngine On
        RewriteCond %{HTTPS} off [OR]
        RewriteCond %{HTTP_HOST} !^www\. [NC]
        RewriteCond %{HTTP_HOST} ^(?:www\.)?(.+)$ [NC]
        RewriteRule ^ https://www.%1%{REQUEST_URI} [L,NE,R=301]
</VirtualHost>

<VirtualHost *:443>
        ServerAdmin fol.pavel@gmail.com
        ServerName zoumi.cz
        RewriteEngine on
        Redirect permanent / https://www.zoumi.cz
</VirtualHost>

<VirtualHost *:443>
        ServerName www.zoumi.cz
        SSLEngine on
        SSLCertificateFile
/etc/letsencrypt/live/www.zoumi.cz/fullchain.pem
        SSLCertificateKeyFile
/etc/letsencrypt/live/www.zoumi.cz/privkey.pem
        Include /etc/letsencrypt/options-ssl-apache.conf

        ServerAdmin fol.pavel@gmail.com

        ProxyPreserveHost on
        ProxyPass / http://127.0.0.1:9090/
        ProxyPassReverse / http://127.0.0.1:9090/

        <Proxy http://localhost:9090/*>
                Require all granted
        </Proxy>
        <Location /shutdown>
                Order deny,allow
                Deny from all
        </Location>
</VirtualHost>
```

Příloha D – *Konfigurační soubor pro webovou stránku ci.zoumi.cz/jenkins*

```
<VirtualHost *:80>
        ServerName ci.zoumi.cz
        ServerAdmin fol.pavel@gmail.com
        RewriteCond %{HTTP_HOST} ci\.zoumi\.cz [NC]
        RewriteCond %{SERVER_PORT} 80
        RewriteRule ^(.*)$ https://ci.zoumi.cz/$1 [R,L]
        Redirect permanent /jenkins https://ci.zoumi.cz/jenkins
</VirtualHost>

<VirtualHost *:443>
        ServerName ci.zoumi.cz
        SSLEngine on
        SSLCertificateFile
/etc/letsencrypt/live/ci.zoumi.cz/fullchain.pem
        SSLCertificateKeyFile
/etc/letsencrypt/live/ci.zoumi.cz/privkey.pem
        Include /etc/letsencrypt/options-ssl-apache.conf

    ServerAdmin  fol.pavel@gmail.com
    ProxyRequests     Off
    ProxyPreserveHost On
    AllowEncodedSlashes NoDecode
    <Proxy *>
        Order deny,allow
        Allow from all
    </Proxy>
    ProxyPass         /jenkins  http://localhost:8080/jenkins
nocanon
    ProxyPassReverse  /jenkins  http://localhost:8080/jenkins
    ProxyPassReverse  /jenkins  http://ci.zoumi.cz/jenkins
    RequestHeader set X-Forwarded-Proto "https"
    RequestHeader set X-Forwarded-Port "443"
</VirtualHost>
```

Příloha E – *Pipeline script pro zoumi-onCommit*

```
node {
    timestamps {
        stage('Preparation') {
            git 'https://github.com/f0lik/zoumi.git'
        }
        stage('Build') {
            if (isUnix()) {
                sh "mvn -Dmaven.test.failure.ignore clean
compile package"
            } else {
                bat(/mvn -Dmaven.test.failure.ignore clean
compile package/)
            }
        }
        stage('Results') {
            junit '**/target/surefire-reports/TEST-*.xml'
            archive 'target/*.jar'
        }
    }
}
```

Příloha F – *Pipeline script pro zoumi-integrationTests*

```
node {
    timestamps {
        stage('Preparation') {
            git 'https://github.com/f0lik/zoumi.git'
        }
        stage('Run') {
            if (isUnix()) {
                sh "mvn -Dmaven.test.failure.ignore clean
compile package -DskipUTs=true"
            } else {
                bat(/mvn -Dmaven.test.failure.ignore clean
compile package -DskipUTs=true/)
            }
        }
        stage('Verify') {
            if (isUnix()) {
                sh "mvn -Dmaven.test.failure.ignore verify -
DskipUTs=true -Ddbuser=user -Ddbpasswd=pass -
Dspring.profiles.active=ittest"
            } else {
                bat(/mvn -Dmaven.test.failure.ignore verify -
DskipUTs=true -Ddbuser=user -Ddbpasswd=pass -
Dspring.profiles.active=ittest"/)
            }
        }
        stage('Results') {
            junit '**/target/failsafe-reports/TEST-*.xml'
        }
    }
}
```

Příloha G – *Freestyle script pro zoumi-deployFreestyle*

```
mvn -DskipTests=true clean compile package
cp target/*.jar /var/www/zoumi-production/zoumi.jar
cd /var/www/zoumi-production
JENKINS_SERVER_COOKIE=dontkill
/var/www/zoumi-production/start.sh
echo 'Your app is now available on http://www.zoumi.cz/'
```