FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Studie proveditelnosti vylepšení řešení jazykové lokalizace webových aplikací v Pythonu
Student:	Petr Kučera
Vedoucí:	Ing. Miroslav Hrončok
Studijní program:	Informatika
Studijní obor:	Informační systémy a management
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

1. Vypracujte studii proveditelnosti vylepšení řešení jazykové lokalizace webových aplikací v jazyce Python, která dá podklad k tomu, zda je takové vylepšení realizovatelné. Součástí studie proveditelnosti je analýza alespoň tří již existujících řešení pro jazykovou lokalizaci webových aplikací v jazyce Python, stanovení měřitelných kritérií a porovnání jednotlivých stávajících řešení na praktických příkladech. Na základě provedené analýzy vyhodnoťte nejlepší dosavadní řešení.
2. Podle výsledků na teoretické úrovni navrhněte úplně nové řešení, případně obohaťte nejvhodnější nalezené existující řešení o novou funkcionalitu, která danému řešení významně přidá hodnotu.
3. Vlastní řešení nemusí být implementováno, ale z návrhu musí být patrné, proč je toto řešení vhodnější a jakým způsobem jej je možné implementovat. Volitelně doplňte návrh implementací na úrovni prototypu.
4. Závěrem práce bude shrnutí všech analýz a odůvodněné vysvětlení, zda je vylepšení realizovatelné.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 16. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Studie proveditelnosti vylepšení řešení jazykové lokalizace webových aplikací v Pythonu

Petr Kučera

Katedra softwarového inženýrství
Vedoucí práce: Ing. Miroslav Hrončok

30. července 2020

Poděkování

Nejprve bych rád poděkoval svému vedoucímu Ing. Miroslavu Hrončokovi za jeho trpělivost a cenné rady při vypracování této práce. Mnohé díky patří Fakultě informačních technologií ČVUT, za znalosti a schopnosti, které mi poskytla. Dále bych chtěl poděkovat své rodině, a všem kteří mne během psaní této práce podporovali.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 30. července 2020 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2020 Petr Kučera. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Kučera, Petr. *Studie proveditelnosti vylepšení řešení jazykové lokalizace webových aplikací v Pythonu.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

V práci je provedena studie proveditelnosti vylepšení lokalizačních řešení v jazyce Python. Práce se nejprve zabývá seznámením čtenáře s nutnými pojmy v problematice studie proveditelnosti a softwarové lokalizace v jazyce Python. Následně je provedena důkladná analýza existujících řešení lokalizace v jazyce Python. Z výstupu této analýzy je určeno, jaké řešení je nejlepší. Po vybrání nejlepšího řešení je zhodnoceno, zda je toto řešení dostačující a je rozhodnuto, jaká významná funkcionalita mu chybí. V následující části je předložen návrh implementace, který slouží jako nastínění finální podoby přidaných funkcionalit a případně jako dokumentace pro budoucí design a implementaci. Přidání funkcionality do řešení je strategicky, ekonomicky a právně analyzováno. Ve výsledku práce poskytuje všestrannou analýzu potřebnou pro doložení, že je vylepšení lokalizačních řešení v jazyce Python proveditelné.

Klíčová slova studie proveditelnosti, lokalizace softwaru, internacionalizace softwaru, projektová správa, Weblate, open source, Python

Abstract

Feasibility study of improvement to localization solutions in Python is performed in this work. The work first deals with acquainting the reader with the necessary concepts in the field of feasibility study and software localization in Python. Subsequently, a thorough analysis of existing localization solutions in Python is performed. From the output of this analysis it is determined which solution is the best. After selecting the best solution, it is evaluated whether this solution is sufficient and it is decided what significant functionality it lacks. In the following part, an implementation proposal is presented, which serves as an outline of the final form of the added functionalities and possibly as documentation for future design and implementation. Adding functionality to the solution is strategically, economically and legally analyzed. As a result, this thesis provides a comprehensive analysis needed to demonstrate that improvements to localization solutions in Python are feasible.

Keywords feasibility study, software localization, software internationalization, project management, Weblate, open source, Python

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Úvod

V současné době je globalizace – hlavně zásluhou internetu – na nejvyšší úrovni, na jaké kdy byla. Znamená to, že téměř v každém koutu světa je možné se spojit s někým na protější straně Země. Ovšem v každém koutu Země se hovoří jiným jazykem a aplikace v anglickém jazyce nemá zdaleka tak velký dosah, jako aplikace přeložená do mateřského jazyka uživatelů. Proto je v rámci propojení světa důležité řešit problém lokalizace.

V posledních letech se softwarová lokalizace posunula z funkcionality, kterou vývojáři přidávali do svého softwaru sami, na multi-miliardové profesionální business odvětví [1].

Lokalizace – společně s internacionalizací – nyní patří do jednoho z klíčových aspektů zajištění úspěchu produktu na mezinárodní úrovni. Z tohoto důvodu se kvalitní lokalizace stala důležitým problémem pro mnoho firem.

U mnoha aplikací je lokalizace jediný důvod, proč nemohou být efektivně vydány do mnoha zemí najednou.

První část práce je věnována teoretické rešerši problematiky studie proveditelnosti a softwarové lokalizace v jazyce Python a frameworku Django. Část slouží jako teoretický podklad pro zbytek práce.

V druhé části je provedena důkladná analýza existujících řešení a vyhodnocení těchto řešení na předem definovaných aspektech. Z analýzy vyplyne nejlepší dosavadní řešení a určí následující postup práce.

V následujících částech práce jsou prozkoumány zbylé aspekty studie proveditelnosti. Obsahují tedy technickou, strategickou, ekonomickou a právní analýzu. Technická analýza je realizována konceptuálním návrhem pro lepší imitaci finální podoby produktu.

Cíl práce

Cílem této práce je vypracování studie proveditelnosti vylepšení řešení jazykové lokalizace webových aplikací v jazyce Python. Tato studie dá podklad k zjištění, zda je vylepšení realizovatelné a vyplatí-li se do něj investovat zdroje.

V práci musí být analyzovány – podle stanovených měřitelných kritérií – nejméně 3 dosud existující řešení na praktických příkladech. Na základě této analýzy bude vyhodnoceno nejlepší dosavadní řešení.

Podle výsledků analýzy je dalším cílem na teoretické úrovni navrhnout úplně nové řešení, případně obohatit nejvhodnější nalezené existující řešení o novou funkcionalitu, která danému řešení významně přidá hodnotu. Samotné řešení nemusí být implementováno.

Pro vybrané řešení musí být provedeny všechny analýzy, které jsou nutné pro rozhodnutí, zda je řešení proveditelné.

Celkově musí práce obsahovat dostatečný průzkum, na kterém bude jasně vidět, zda je řešení realizovatelné, či ne.

Kapitola 1

Technologie a pojmy použité v práci

1.1 Studie proveditelnosti

Studie proveditelnosti (anglicky feasibility study) je hodnocení praktičnosti určitého projektu (jedinečný soubor procesů plánovaných pro dosažení konkrétního cíle). Objektivně a nezaujatě zohledňuje všechny relevantní faktory – ekonomické, technické i právní. Po přečtení studie by čtenář měl mít dobrou znalost, zda je možné projekt realizovat ještě před investováním peněz nebo času [2].

Někdy bývá studie proveditelnosti označována jako technicko-ekonomická studie. Jedná se o dokument, který ze všech realizačně významných aspektů popisuje investiční záměr. Jeho účelem je vyhodnotit kompletně realizační možnosti a určit realizovatelnost investičního projektu, jakožto i poskytnout veškeré podklady pro samotné investiční rozhodnutí [3].

Struktura studie proveditelnosti není pevně stanovená, ovšem ve většině zdrojů se vyskytuje několik hlavních částí [3, 2, 4, 5]. V následujících sekcích budou hlavní části studie popsány.

1.1.1 Popis projektu

Toto je první krok při přípravě studie proveditelnosti vývoj softwaru. V této sekci se musí přesně definovat záměr projektu, kterým se následně bude zabývat celá studie proveditelnosti [4].

1.1.2 Analýza existujících řešení

Zkoumání trhu po existujícím produktu, službě, či firmě, která řeší analyzovaný projekt [2]. V sekci by měly být existující řešení detailně zhodnoceny ve všech významných aspektech.

1.1.3 Technická analýza

Tato sekce popisuje jak by mohl být projekt realizovaný. Slouží jako hlavní podklad pro časový odhad [4].

1.1.4 SWOT analýza

SWOT analýza je důležitou součástí studie proveditelnosti. Je schopna identifikovat podmínky, potenciál a problémy souvisejících aspektů, které můžou ovlivnit řadu rozhodnutí [6].

SWOT analýza je jedním z nejjednodušších způsobů, jak provést strategickou analýzu. SWOT je anagram pro 4 klíčové elementy:

- **Silné stránky** – „Strengths", ovlivnitelné faktory analyzovaného řešení, které by mohly přinést výhodu oproti ostatním řešením.
- **Slabé stránky** – „Weaknesses", opak silných stránek. Ovlivnitelné faktory, kterým lze předejít, či je minimalizovat.
- **Příležitosti** – „Opportunities", neovlivnitelné faktory z vnějšího prostředí, které by mohly přinést řešení výhodu.
- **Rizika** – „Threats", přesný opak příležitostí – faktory, které nemohou být ovlivněny, ale mohly by vyústit v neúspěch řešení [7].

Účelem je upřesnit cíle podniku nebo projektu a identifikovat vnitřní a vnější faktory, které jsou pro dosažení těchto cílů příznivé a nepříznivé. Uživatelé SWOT analýzy se ptají a odpovídají na otázky, aby vytvořili smysluplné informace pro každou z kategorií *SWOT*. Takto se identifikují konkurenční výhody projektu. SWOT analýza je popisována jako osvědčený a dobrý nástroj pro strategickou analýzu [8].

1.1.5 Ekonomická analýza

V ekonomické části studie proveditelnosti jsou analyzovány náklady a přínosy projektu. V rámci analýzy se zkoumá, jaké budou náklady na návrh a vývoj, provozní náklady atd. Poté je analyzováno, jak bude projekt přínosný [5].

1.1.6 Právní analýza

Právní část studie proveditelnosti analyzuje projekt z právního hlediska. To zahrnuje analýzu právních překážek při implementaci projektu, zákonů o ochraně dat, licencí, autorských práv atd. Celkově lze říci, že právní analýza zkoumá, zda je navržený projekt v souladu s právními a etickými požadavky [5].

1.1.7 Nastínění struktury této studie proveditelnosti

1. Popis projektu
2. Analýza existujících řešení
3. Konceptuální návrh
4. Strategická analýza
   - 4.1. SWOT analýza
   - 4.2. Marketingová strategie
5. Ekonomická a právní analýza
   - 5.1. Odhad práce
   - 5.2. Odhad ceny implementace
   - 5.3. Provozní náklady
   - 5.4. Přínos projektu
   - 5.5. Právní analýza

1.2 T-shirt odhady

Do češtiny jsou doslova přeložené jako „Trikové odhady."Většinou se ale využívá anglické jméno. Jedná se o metodu používanou při odhadu práce.

T-shirt odhady používají velikosti S, M, L, XL (někdy i vyšší) jako způsob, jak odhadnout časovou náročnost úkolu. Cílem je projít všechny jednotky práce v týmu a nechat členy týmu vložit je do jedné z definovaných velikostí. Velikosti triček lze definovat dle potřeb týmu a rozsahu úkolů. Obvykle jsou definovány tabulkou.

Pokud je úkol příliš velký na to, aby mu byla přiřazena nějaká z velikostí, je zapotřebí se pokusit o rozdělení úkolu na více menších úkolů. Pokud nelze úkol odhadnout a jeho odhad by byl příliš abstraktní, automaticky by měl být odhadnut jako největší velikost [9].

1.3 Projektová správa

Správa projektu je způsob vedení týmu k dosažení cílů a splnění kritérií úspěchu v předem stanoveném čase. Primární výzvou řízení projektu je dosažení všech cílů projektu při splnění daných omezení [10]. Tyto informace jsou obvykle popsány v projektové dokumentaci vytvořené na začátku vývojového procesu. Primárními omezeními projektu jsou rozsah, čas, kvalita a rozpočet [11]. Při snaze zlepšení jednoho z omezení se tak většinou zlepšení provede na úkor omezení ostatních. Například pro snížení času je zapotřebí omezit rozsah, snížit kvalitu nebo zvýšit rozpočet, případně kombinaci těchto možností.

1.4 UML

Unifikovaný modelovací jazyk (UML) je standardizovaný jazyk pro vytváření návrhů softwaru. UML lze použít k vizualizaci, specifikaci, konstrukci a dokumentování artefaktů softwarově náročného systému [12]. UML je otevřený standard, není to proprietární jazyk. Jako takový, může UML používat kdokoli a kdekoli [13]. UML zahrnuje tři druhy stavebních bloků:

- věci,
- vztahy a
- diagramy.

Věci jsou abstrakce z reálného světa, které jsou primárními objekty v modelu. Vztahy tyto věci spojují. Diagramy seskupují věci do sbírek. V UML jsou čtyři druhy věcí:

- strukturální,
- behaviorální,
- seskupovací a
- anotační.

Tyto věci jsou základní objektově orientované stavební bloky UML. Jsou používány k vytváření dobrých modelů [12].

1.5 Případy užití

Případy užití (anglicky use cases) jsou v softwarovém a systémovém inženýrství myšleny jako seznam akcí nebo kroků událostí k dosažení cíle. Obvykle jsou kroky definovány interakcí mezi rolí (v UML nazývaný herec) a systémem [14]. Rolí může být člověk nebo jiný externí systém. V systémovém inženýrství se případy užití používají na vyšší úrovni než v softwarovém inženýrství, často představují mise nebo cíle stakeholderů [15].

Modelování případů užití je často spojováno s UML, i když případy užití byly zavedeny dříve než UML samotné [16].

1.6 Wireframe

Wireframe je zjednodušený pohled na obrazovku navrhované aplikace (hlavně webové a mobilní aplikace), bez jakýchkoliv estetických úprav, až na absolutní minimum [17]. Wireframy zachycují základní kostru aplikace v dokumentu, který slouží jako reference pro vizuální design a implementaci aplikace.

Wireframy mohou být vytvořeny v různých úrovních detailu. Většinou jsou wireframy vytvářeny pomocí velmi jednoduchých bezbarvých nákresů, postrádajících jakýkoliv design. Pokud ale wireframy vznikají pro již existující aplikaci, často se pro návrh využije existující design [18].

Wireframy mohou být klikatelné a mohou velmi dobře imitovat finální uživatelskou interakci. Také se dají velmi jednoduše změnit na základě obdržené zpětné vazby. Proto mohou být použity při prezentaci aplikace stakeholderům, ještě před začátkem vývoje [19].

1.7 Python

Python je interpretovaný, univerzální, vyšší programovací jazyk. Byl vytvořen Guido van Rossumem a poprvé zveřejněn v roce 1991. Design Pythonu zdůrazňuje čitelnost kódu díky jeho významnému používání whitespace znaků. Cílem architektury jazyka a objektově orientovaného přístupu je pomoci programátorům vytvářet čistý a logický zdrojový kód pro malé i velké projekty [20].

Python je dynamicky psaný a obsahuje garbage-collector. Podporuje několik programovacích paradigmat, včetně strukturovaného (zejména procedurálního), objektově orientovaného a funkčního programování. Python je díky své obsáhlé standardní knihovně často popisován jako jazyk „včetně baterií“. Python má rozhraní pro mnoho systémových volání a knihoven a je rozšiřitelný v C nebo C++ [21].

1.8 Webový framework

Webový framework nebo také webový aplikační framework je softwarový framework, který je navržen pro podporu vývoje webových aplikací včetně webových služeb, webových zdrojů a webových API. Webové frameworky poskytují standardní způsob vytváření a nasazování webových aplikací na webu. Cílem webových frameworků je automatizovat režijní náklady spojené s běžnými činnostmi prováděnými při vývoji webových aplikací. Mnoho webových frameworků poskytuje například knihovny pro přístup k databázi, šablonové frameworky a správu relací. Často podporují opětovné použití kódu [22]. Přestože se často zaměřují na vývoj dynamických webových stránek, lze je použít i na statické webové stránky [23].

1.9 Django framework

Django je open source webový framework, který dodržuje MVC (v Djangu model-template-view) softwarovou architekturu [24]. Je spravován americkou nezávislou neziskovou organizací Django Software Foundation (DSF) a byl vytvořen roku 2003.

Hlavním cílem Djanga je usnadnění vytváření komplexních databázově řízených webových stránek. Framework zdůrazňuje znovupoužitelnost a „připojitelnost" komponent, méně kódu, nízkou vazbu, rychlý vývoj a princip DRY (don't repeat yourself) [25]. Python je používán skrze celý framework, i pro nastavení a datové modely. Django framework také volitelně poskytuje administrativní CRUD (create, read, update, delete) rozhraní, které je generováno dynamicky pomocí introspekce a konfigurováno pomocí admin modelů.

Jedny z nejznámějších aplikací využívající Django framework jsou například Instagram [26], Mozilla Infrastructure [27] a The Washington Times [28].

1.10 Systém řízení překladů

Anglicky Translation management system (dále TMS), někdy známý jako systém řízení globalizace, je software určený k automatizaci a zefektivnění mnoha součástí lidského procesu překladu. Systém se snaží automatizovat všechnu práci, která je repetitivní a je možné ji udělat pomocí softwaru. V ideálním případě zbývá pouze „kreativní" práce překladatelů a revizorů. TMS většinou obsahuje minimálně dva typy technologií – technologie podporující správu procesu pro automatizaci workflow a lingvistické technologie napomáhající překladatelům [29].

V typickém TMS se technologie řízení procesů používá ke sledování obsahu zdrojového jazyka pro změny a směrování překladů k různým překladatelům a revizorům. Tito překladatelé a revizoři se mohou nacházet po celém světě a obvykle mají přístup k TMS přes internet.

1.11 Internacionalizace (I18N)

Internacionalizace je operace, pomocí které je program – či balíček programů – schopen podporovat více jazyků. Jedná se o generalizační proces, při kterém jsou programy odvázány od volání pouze statických anglických řetězců nebo jiných anglických zvyklostí a místo toho jsou implementovány obecné způsoby, jak udělat to samé. Vývojáři programů mohou k internacionalizaci svých programů použít různé techniky. Některé z těchto technik byly standardizovány. GNU gettext nabízí jeden z těchto standardů [30, 31].

1.12 Lokalizace (L10N)

Lokalizací se rozumí operace, která souboru již internacionalizovaných programů poskytuje všechny potřebné informace, aby se mohl přizpůsobit tak, aby upravil svůj vstup a výstup způsobem, který je správný pro zvolené jazyky a kulturní zvyky. Jedná se o proces individualizace, pomocí kterého se generické metody již implementované v internacionalizovaném programu používají

specifickým způsobem. Programovací prostředí dává programátorům k dispozici několik funkcí, které tuto runtime konfiguraci umožňují. Formální popis specifické sady kulturních návyků pro určitou zemi, spolu se všemi souvisejícími překlady cílenými do stejného jazyka, se nazývá locale pro tento jazyk nebo zemi.

Uživatelé dosáhnou lokalizace programů nastavením správných hodnot na speciální proměnné prostředí před spuštěním těchto programů a určením, které locale by mělo být použito [30, 31].

1.13 GNU gettext

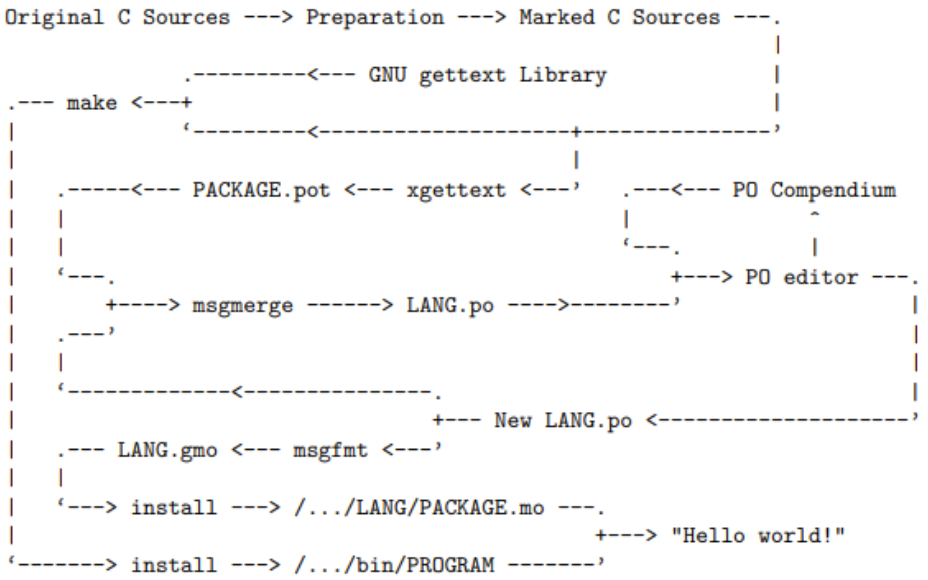
Balíček GNU gettext nabízí programátorům, překladatelům a dokonce uživatelům dobře integrovanou sadu nástrojů a dokumentaci. Konkrétně, služby GNU gettext jsou sada nástrojů, které poskytují framework, ve kterém mohou jiné bezplatné balíčky produkovat vícejazyčné programy. Tyto nástroje zahrnují:

- Sada konvencí o tom, jak by se programy měly psát na podporu katalogů zpráv.
- Konvence pro strukturu a pojmenování adresářů a souborů katalogů zpráv.
- Runtime knihovnu podporující načítání přeložených zpráv.
- Knihovnu podporující analýzu a vytváření souborů obsahujících přeložené zprávy.
- Speciální režim pro Emacs1, který pomáhá připravovat tyto sady a aktualizovat je.

GNU gettext je navržen tak, aby minimalizoval dopad internacionalizace na zdroje programu a udržoval tento dopad co nejmenší a stěží viditelný [30].

1.13.1 Jak GNU gettext funguje

V ukázce je nastíněno jak GNU gettext funguje na programy napsané v jazyce C. Diagram 1.1 shrnuje relaci mezi soubory používanými GNU gettextem a nástroji provádějícími akce na těchto souborech. Následuje popis, který pomůže s pochopením diagramu.



Obrázek 1.1: GNU gettext diagram. [30]

Prvním krokem je označení řetězců určených k překladu přímo ve zdrojovém kódu C. Tato zdlouhavá práce se dá ulehčit například pomocí emacs PO (portable object) módu. Je doporučeno řetězce označit přímo při psaní programu.

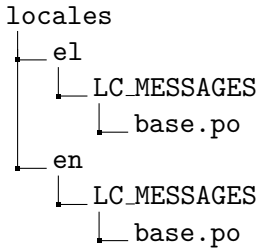
Potom, co jsou všechny zdrojové kódy upravené, je program `xgettext` použit pro nalezení a extrakci všech překladových řetězců. Poté ze všech řetězců vytvoří PO template soubor (dále POT). POT soubor obsahuje všechny originální řetězce. Nejsou orientované na žádný partikulární jazyk, slouží jako šablona (anglicky template).

Jelikož nejdříve není žádná verze PO souboru, je možné přeskočit krok `msgmerge`. Později ale tento program slouží pro spojení již vytvořených PO souborů zaměřených na určitý jazyk s novou verzí POT souboru, která může obsahovat aktualizované zdrojové řetězce.

Nyní přichází samotný překlad řetězců. Překlad je přenechán překladatelům. Při přidávání přeložených řetězců do PO souboru `lang.po` je zapotřebí respektovat formát PO souboru. Pro zajištění formátu PO souborů se dá využít jednoho z mnoha PO editorů.

Poté, co jsou PO soubory kompletní, je program `msgfmt` použit pro přetvoření PO souborů do strojově orientovaného (MO) formátu, který přináší efektivní vyhledávání překladů pomocí programů balíčku, kdykoli jsou za běhu potřeba.

Nakonec jsou upravené a označené zdrojové kódy C kompilovány a propojeny s knihovnou GNU gettext, obvykle prostřednictvím operace make. MO soubory by také měly být adekvátně nainstalovány. Pokud jsou nastaveny správné proměnné prostředí, C program by se měl automaticky lokalizovat při každém spuštění [30]. Adresářová struktura PO souborů by měla vypadat následovně:



Obrázek 1.2: Adresářová struktura PO souborů. [32]

1.13.2 Překladové katalogy

Jindy také nazývány překladové soubory, soubory zpráv (message file), katalogy zpráv, gettext katalogy, či gettext PO soubory. V této práci bude nadále používán termín *překladový katalog*.

Překladový katalog je určený k překladu zdrojových řetězců do jiných jazyků. Soubor má tedy takovou strukturu, aby byl snadno pochopitelný člověkem, který zastává roli překladatele.

1.13.2.1 Formát překladového katalogu

PO soubor se skládá z mnoha položek, z nichž každá je vztah mezi jedním původním nepřeloženým řetězcem – msgid – a jeho odpovídajícím překladem – msgstr. Všechny položky v daném souboru PO se obvykle vztahují k jednomu projektu a všechny překlady jsou vyjádřeny v jednom cílovém jazyce [33]. Nad každou položkou je vždy reference, odkazující na přesné místo, kde se řetězec ve zdrojovém kódu vyskytuje. Volitelně může být uveden msgctxt (udávající kontext k překladu), komentář pro překladatele, flag a další. Na obrázku 1.1 je vidět schematická struktura jedné položky PO souboru. Dále je možné specifikovat hlavičky pro zajištění více informací o locale. Lze vidět na obrázku 1.2.

```
#  translator-comments
#. extracted-comments
#: reference...
#, flag...
#| msgid previous-untranslated-string
msgctxt context
msgid untranslated-string
msgstr translated-string
```

Výpis kódu 1.1: Ukázka struktury PO souboru.

```
# My App.
#
msgid ""
msgstr ""
"Project-Id-Version: 1.0\n"
"POT-Creation-Date: 2018-01-28 16:47+0000\n"
"PO-Revision-Date: 2018-01-28 16:48+0000\n"
"Last-Translator: me <johndoe@example.com>\n"
"Language-Team: Czech <yourteam@example.com>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Generated-By: pygettext.py 1.5\n"
#: main.py:5
msgid "Hello world"
msgstr "Ahoj světe"
#: main.py:6
msgid "This is a translatable string"
msgstr "Toto je přeložitelný řetězec"
```

Výpis kódu 1.2: Ukázka hlaviček PO souboru.

1.14 Internacionalizace v Pythonu pomocí GNU gettext

Internacionalizace v Pythonu může být řešena více způsoby. Jedním z nich – který využívá i Django framework – je internacionalizace pomocí GNU gettext frameworku. Více o tomto frameworku je v sekci 1.13.

Při využití GNU gettext se při internacionalizaci postupuje dle standardizovaného způsobu určeného v dokumentaci GNU gettext. Je zapotřebí:

1. Připravit program nebo modul pomocí označení řetězců určených k překladu.

2. Spustit sadu nástrojů nad označenými soubory pro vytvoření překladových katalogů.

3. Provést překlady v překladových katalozích.

4. Použít `gettext` modul, aby se z překladových řetězců vygenerovaly potřebné soubory [34].

Jednotlivé kroky do detailu popsané nebudou, jelikož internacionalizace Django frameworku je také postavená na GNU gettext a je detailně popsaná v sekci 1.15.

1.15 Internacionalizace a lokalizace v Django frameworku

1.15.1 Překladové katalogy

V Django frameworku se tímto termínem míní textové soubory, které reprezentují překlady do jednoho jazyka. Zastávají stejnou funkcionalitu a mají stejnou schematickou strukturu jako překladové katalogy rozebrané v sekci 1.13.2.

1.15.2 Internacionalizace zdrojového kódu

Internacionalizace v Djangu se aktivuje jednoduše pomocí úpravy projektového souboru `settings.py` (soubor s nastavením projektu), kde se nastaví konstanta `USE_I18N=True`.

Poté se musí všechny texty, které jsou určeny k překladu, označit příslušnými funkcemi. Ukázku užití funkce `gettext()` a korespondující část překladového katalogu lze vidět na ukázkách kódu 1.3 a 1.4.

Pokud je překlad zapotřebí pluralizovat v závislosti na proměnné, dá se použít funkce `ngettext()`, u které se mohou definovat dva plurály v závislosti na zvolené proměnné. Ukázáno na výpisu kódu 1.5.

Pokud je v jazyce, kam se text překládá, více plurálů (v češtině jsou například 4), pak je možné v hlavičce Plural-Form překladového katalogu specifikovat počet plurálů a zároveň rovnici na specifikování plurálu podle čísla. Ukázka výpisu kódu 1.7. Korespondující část překladového katalogu k pluralizaci je vidět na výpisu kódu 1.6.

Na ukázce kódu 1.8 je vidět důležitá funkce `pgettext()`, pomocí které je možné přidat k textu kontext, který je pro překladatele nutný v případě, že překládaný text má ve zdrojovém jazyce více významů. Korespondující část překladového katalogu je vidět na výpisu kódu 1.9.

```python
from django.http import HttpResponse
from django.utils.translation import gettext

def my_view(request):
    output = gettext("Welcome to my site.")
    return HttpResponse(output)
```

Výpis kódu 1.3: Ukázka funkce `gettext()`. [35]

```
#: path/to/python/module.py:23
msgid "Welcome to my site."
msgstr ""
```

Výpis kódu 1.4: Ukázka překladových stringů v katalogu. [35]

```python
from django.http import HttpResponse
from django.utils.translation import ngettext

def hello_world(request, count):
    page = ngettext(
        'there is %(count)d object',
        'there are %(count)d objects',
    count) % {
        'count': count,
    }
    return HttpResponse(page)
```

Výpis kódu 1.5: Ukázka funkce `ngettext()`. [35]

```
#: path/to/python/module.py.html:16
#, python-format
msgid ""
"\n"
"          <h2>%there is %(count)d object</h2>\n"
msgid_plural ""
"\n"
"          <h2>there are %(count)d objects</h2>\n"
msgstr[0] ""
msgstr[1] ""
msgstr[2] ""
msgstr[3] ""
```

Výpis kódu 1.6: Ukázka pluralizace v katalogu. [35]

```
"Plural-Forms: nplurals=4; plural=(n == 1 && n % 1 == 0) ? 0 : (n >= 2 && n "
"<= 4 && n % 1 == 0) ? 1: (n % 1 != 0 ) ? 2 : 3;\n"
```

Výpis kódu 1.7: Ukázka nastavení pluralizace v katalogu. [35]

```
from django.utils.translation import pgettext

month = pgettext("month name", "May")
```

Výpis kódu 1.8: Ukázka funkce `pgettext()`. [35]

```
msgctxt "month name"
msgid "May"
msgstr ""
```

Výpis kódu 1.9: Ukázka kontextu v katalogu.

1.15.3 Internacionalizace šablon

Django HTML šablony (templates), podle kterých se generuje struktura webové aplikace, se internacionalizují podobným způsobem označení textů. V šabloně stačí importovat modul `i18n` a jednoduché texty určené k překladu obalit pomocí funkce `trans` (ukázka kódu 1.10). Složitější texty, které obsahují HTML značky nebo jsou na více řádků, se musí obalit funkcí `blocktrans` (ukázka kódu 1.10). Na ukázce 1.11 je vidět korespondující část překladového katalogu.

```
{% comment %}Translators: View verb{% endcomment %}
{% trans "View" %}

{% comment %}Translators: Short intro blurb{% endcomment %}
<p>{% blocktrans %}A multiline translatable
literal.{% endblocktrans %}</p>
```

Výpis kódu 1.10: Internacionalizace Django šablon. [35]

1.15.4 Komentáře pro překladatele

Jak je vidět v sekci 1.13.2.1, k překladům se dají připnout komentáře pro překladatele. V šablonách toho docílíme pomocí funkce `comment`, viz kód 1.10.

Na ukázce 1.12 je vidět, jak docílíme komentáře v Python kódu. Stačí vytvořit komentář začínající klíčovým slovem „Translators". Zobrazení komentáře v překladovém katalogu je ukázáno na výpisu kódu 1.1.

```
#. Translators: View verb
# path/to/template/file.html:10
msgid "View"
msgstr ""

#. Translators: Short intro blurb
# path/to/template/file.html:13
msgid ""
"A multiline translatable"
"literal."
msgstr ""
```

Výpis kódu 1.11: Internacionalizace šablon v překladovém katalogu. [35]

```
def my_view(request):
# Translators: This message appears on the home page only
output = gettext("Welcome to my site.")
```

Výpis kódu 1.12: Ukázka komentářů pro překladatele. [35]

1.15.5 Generování překladových katalogů

Po označení všech textů, které jsou zapotřebí přeložit, je možné vygenerovat překladové soubory. Toho se dá docílit pomocí příkazu `makemessages` (ukázka použití ve výpisu kódu 1.13). Příkaz projde celý adresářový strom projektu a najde všechny řetězce určené k překladu. Poté vytvoří nebo aktualizuje překladový katalog v nastaveném locale adresáři.

Po změně překladových katalogů je zapotřebí je zkompilovat pomocí příkazu `compilemessages` (ukázka použití ve výpisu kódu 1.14). Příkaz `compilemessages` vytvoří soubor s příponou `.mo` do příslušného locale adresáře. Tyto soubory jsou využité vestavěnou GNU gettext podporou pro dosazení správných řetězců podle nastaveného jazyka [36]. Více o gettext v sekci 1.13.

```
django-admin makemessages --locale=pt_BR
```

Výpis kódu 1.13: Ukázka užití příkazu makemessages pro vytvoření brazilských překladových katalogů.

```
django-admin compilemessages --locale=pt_BR
```

Výpis kódu 1.14: Ukázka užití příkazu compilemessages pro zkompilování brazilských překladových katalogů.

1.15.6 Lokalizace

Lokalizace je provedena pomocí manuálního přeložení vygenerovaných překladových souborů.

Kapitola 2

Analýza existujících řešení

2.1 Popis zkoumaného projektu

Cílem tohoto projektu je vytvořit nové řešení lokalizace pro webové aplikace v jazyce Python, případně vylepšit nejlepší dosavadní řešení. Projekt má záměr vylepšit dosavadní lokalizační řešení a přispět tak globalizaci a propojení světa. Projekt není realizován se záměrem navratitelnosti investice, ale s úmyslem přispět dosavadnímu stavu lokalizačních řešení.

2.2 Vybraná řešení

Pro různé frameworky existuje mnoho lokalizačních řešení. Z toho důvodu se v této práci porovnávají pouze řešení specifická pro Django framework, aby bylo možné přímočařejší srovnání.

V této práci nazývané Django basic, je řešení, které využívá pouze nástroje, které jsou v Django frameworku již obsažené. Slouží jako kontrolní řešení pro srovnání s ostatními. Dále jsou pak hodnocené řešení Rosetta a Django translation manager, dvě open-source řešení, která se mohou použít pouze pro Django framework.

Další 3 analyzovaná řešení jsou obecná, dají se použít na libovolné gettext katalogy: Zanata, Weblate a Lokalise, z nichž první dvě jsou open source produkty.

V práci budou zhodnocena řešení:

- Django basic,
- Rosetta,
- Django translation manager,
- Lokalise,
- Zanata a

- Weblate.

2.3 Internacionalizace u lokalizačních řešení

Všechna lokalizační řešení pracují – a ulehčují práci – s překladovými katalogy vygenerovanými Django frameworkem. Všechna řešení tedy využívají stejnou vestavěnou Django internacionalizaci a do zdrojového kódu internacionalizované apliakce není zapotřebí zasahovat. V analýze je zkoumáno vždy lokalizační řešení samotné, bez ohledu na internacionalizaci.

2.4 Definování hodnocení

2.4.1 Aspekty hodnocení

U každého řešení je hodnoceno 6 aspektů. Výběr aspektů proběhl tak, aby po dokončení hodnocení, bylo řešení posouzeno ze všech významných hledisek. Následuje krátký popis každého aspektu.

Rychlost integrace Ohodnocení rychlosti integrace do již vybudovaného projektu.

Udržitelnost Tento aspekt zohledňuje udržitelnost lokalizačního řešení. Tedy jak časově je náročné udržovat lokalizaci funkční. Zejména nutnost zasahovat znovu do zdrojového kódu projektu, či nastavení lokalizace.

Přístupnost k překladům Hodnocení přístupnosti k překladům zohledňuje, zda jsou překlady dostupné online přes webový prohlížeč či pouze lokálně, a zda je jednoduché novým překladatelům zařídit k překladům přístup. V potaz je také brána přístupnost pro lidi bez zkušeností s programováním, či dokonce se špatnými zkušenostmi s elektronikou.

Užitečné funkcionality navíc V tomto hodnocení se zohledňují ostatní funkcionality, které se nedají považovat za elementární funkcionality nutné k překládání textů. Jedná se zejména o funkce zefektivňující a urychlující proces lokalizace.

Cena a podmínky užití V tomto aspektu je samozřejmě hodnocena cena, ale také podmínky užívání a licence, pod kterou je řešení zveřejněno.

Možnost zasahovat do řešení Zohledňuje možnost vlastních úprav a přizpůsobení řešení.

Možnost revize překladu Hodnocení rozsáhlosti možností revize při procesu překládání textů.

2.4.2 Hodnocení aspektů

Každý aspekt je hodnocen na číselné stupnici od 0 do 5. Čím vyšší číslo je, tím lepší je hodnocené řešení v daném aspektu. Tedy 0 znamená „řešení je v tomto aspektu velmi špatné", naopak hodnocení 5 znamená „řešení je v tomto aspektu velmi dobře provedené až perfektní."

2.5 Testovací projekt

Pro účel testování byla vytvořena jednoduchá Django aplikace, ve které jsou publikovány blogové příspěvky. U každého blogového příspěvku je možnost přidávat komentáře.

Pro důkladné otestování obsahuje aplikace různé typy řetězců. V šablonách se vyskytují statické i dynamické řetězce závisející na počtu položek, či doplnění dat z databáze. V Python kódu jsou pro překlad především statické řetězce, které se vyskytují i v databázových modelech. Aplikace také obsahuje řetězce, u kterých je nutné k překladu znát kontext.

Repozitář s aplikací je k dispozici v přílohách této práce.

2.6 Analýza řešení

2.6.1 Django basic

Popis řešení Termínem Django basic je v této práci nazývána vestavěná funkcionalita Djanga, která se stará o internacionalizaci a částečně i lokalizaci [37]. Lokalizace se v tomto řešení řeší tak, že se vygenerované překladové katalogy upravují manuálně, přímo v kódovém repozitáři. Jakékoliv zlepšení přístupnosti, či přidání funkcionalit se musí řešit vlastní implementací.

Rychlost integrace (5/5) Jelikož je toto řešení v Djangu již vestavěné, je zapotřebí pouze v nastavení projektu specifikovat zvolené jazyky (ukázka v kódu 2.1) a vytvořit překladové katalogy pomocí příkazu `django-admin makemessages`. Django basic tedy v tomto aspektu nastavuje nejvyšší možnou rychlost integrace.

```
        LANGUAGES = (
    ('cs', _('czech')),
    ('en', _('english')),
)
```

Výpis kódu 2.1: Specifikace dostupných jazyků Django projektu.

Udržitelnost (5/5) Podobně jako u aspektu výše, není zapotřebí nic udržovat. Pouze při přidání nového jazyka je potřeba zadat příslušný jazykový kód do nastavení projektu, jako je ukázáno na výpisu kódu 2.1, a vygenerovat nové překladové katalogy. Stejně jako u integrace má Django basic v tomto aspektu perfektní provedení.

Přístupnost k překladům (0/5) Pro tvoření překladů překladatel bez zkušenosti s programováním nemusí sahat přímo do zdrojového kódu. Soubory, do kterých se překlad píše, jsou ale umístěné v repozitáři, kam překladatelé přístup většinou nemají. Nemluvě o tom, když je vyžadován překlad od externích zdrojů, pak je zpřístupnění souborů určených pro překlad velmi složité.

Užitečné funkcionality navíc (0/5) Lokalizace v Django frameworku je velmi minimalistická a neobsahuje absolutně žádnou zlehčující, či užitečnou funkcionalitu pro překladatele. To ovšem přispívá k tvorbě a různorodosti nadstaveb, které poskytují přidanou hodnotu.

Cena a podmínky užití (5/5) Django basic je funkcionalita vestavěná do Django frameworku. Django framework je veřejně dostupný pod 3-bodovou BSD licencí.

Možnost zasahovat do řešení (4/5) Jelikož je Django framework licencovaný pod 3-bodovou BSD licencí, je možné si celý projekt forknout a do kódu zasahovat. Po přidání funkcionality, která přidává hodnotu Django frameworku, je také možnost pomocí pull requestu do projektu přispět.

Možnost revize překladu (0/5) Neposkytuje žádnou možnost označení překladů určených k revizi. Jediná možnost je napsat do překladového katalogu komentáře.

2.6.2 Rosetta

Popis řešení Rosetta je Django aplikace, která zjednodušuje lokalizační proces projektů v Django frameworku. Rosetta aplikace nevytváří žádné databázové tabulky a není tedy na databázi ani závislá. Automaticky nahrává řetězce určené k překladu do webového prostředí vývojového serveru. Po dokončení nových překladů Rosetta automaticky aktualizuje korespondující překladové katalogy a přegeneruje korespondující `.mo` soubory.

Řešení je dostupné online, z [38].

Rychlost integrace (4/5) Implementace Rosetty je velmi jednoduchá a rychlá. Prakticky její implementace zabere téměř stejně času jako Django

basic. Pro nainstalování rosetty je zapotřebí ji stáhnout jako Python package pomocí `pip install django-rosetta`. Následně je zapotřebí ji pouze přidat do nastavení projektu mezi nainstalované aplikace a přidat její URL záznam mezi projektové URL. Integrace Rosetta aplikace do projektu je vidět na výpisech kódu 2.2 a 2.3.

```
from django.conf import settings

if 'rosetta' in settings.INSTALLED_APPS:
    urlpatterns += [
        url(r'^rosetta/', include('rosetta.urls'))
    ]
```

Výpis kódu 2.2: Integrace Rosetta URL.

```
INSTALLED_APPS = (
...
'rosetta',
)
```

Výpis kódu 2.3: Přidání Rosetta aplikace mezi nainstalované aplikace v projektu.

Udržitelnost (5/5) Při dalším vývoji aplikace není třeba ve zdrojovém kódu nic upravovat. Pokud je zapotřebí Rosetta aplikaci aktualizovat, učiní se tak opět pomocí `pip` příkazu.

Přístupnost k překladům (2/5) K překladům se v Rosettě přistupuje přes webovou stránku – na uživatelem zvolené URL – na vývojovém serveru. Po otevření URL uživatel vidí obsah všech překladových katalogů, které Django generuje, přehledně online. Přístup k překladům je skrytý pod autorizací.

Proces lokalizace je možné nastavit tak, že překladatelé budou překládat přímo na produkčním serveru. To ale není ideální řešení z toho důvodu, že provedené překlady by se přepsaly při dalším nasazení nové verze aplikace. Muselo by se tedy nějakým způsobem nastavit zálohování překladových katalogů a `.mo` souborů. Přístup do Rosetty pro externí zdroje je tímto velmi zkomplikován.

Užitečné funkcionality navíc (1/5) Mimo přístupu k překladům přes webový prohlížeč, procentuálního ukazatele kompletnosti překladu celého katalogu a pár filtrů – například na nepřeložené řetězce – neobsahuje Rosetta žádné funkcionality navíc.

Cena a podmínky užití (5/5) Rosetta je open source řešení, které je zdarma. Dílo je dostupné pod licencí MIT.

Možnost zasahovat do řešení (4/5) Jelikož je dílo dostupné pod MIT licencí, je možnost si celé dílo forknout a udělat v něm jakékoliv změny. Pokud by tyto změny mohly přidat hodnotu Rosetta aplikaci, je možnost vytvořit pull request a zažádat o zařazení úprav do originálního projektu.

Možnost revize překladu (0/5) Neposkytuje žádnou možnost označení překladů určených k revizi.

2.6.3 Django translation manager

Popis řešení Django translation manager (dále DTM) je open-source Django aplikace vytvořená českou firmou COEX na základě nespokojenosti s limitami Rosetta aplikace [39].

Podobně jako Rosetta, je založena na v Django frameworku již implementovaných i18n základech a rozšiřuje jeho funkce ve webovém prohlížeči. Hlavní rozdíl oproti aplikaci Rosetta je v tom, že DTM uchovává zálohy překladů v databázi. Navíc je také plně ovladatelný z Django admin prostředí a dají se v něm používat pokročilejší filtry a vytvářet kategorie.

Řešení je dostupné online, z [40].

Rychlost integrace (4/5) Implementace zabere řádově několik minut. Na ukázce 2.5 je možné vidět jak se aplikace stáhne pomocí Python package installeru `pip`. Pak se v nastavení projektu přidá mezi nainstalované aplikace a vyplní se potřebná nastavení (ukázka nastavení na výpisu kódu 2.4).

V poslední řadě se musí aktualizovat schéma databáze, jelikož je na ní aplikace závislá. To lze vidět na ukázce 2.6.

```
INSTALLED_APPS = (
...
'translation_manager',
)

# Required paths to all locale dirs
LOCALE_PATHS = []

# Path to project basedir / workdir - root folder of project
# TRANSLATIONS_BASE_DIR = os.path.dirname(os.path.dirname(__file__))
TRANSLATIONS_BASE_DIR = ''

# Language to display in hint column to help translators
# see translation of string in another language
TRANSLATIONS_HINT_LANGUAGE = ''
```

Výpis kódu 2.4: Potřebná nastavení DTM

```
pip install django-translation-manager
```

Výpis kódu 2.5: Nainstalování DTM aplikace.

```
python manage.py migrate
```

Výpis kódu 2.6: Migrace databáze

Udržitelnost (4,5/5) Po implementaci je zapotřebí při každém přidání nové destinace překladových katalogů přidat cestu k této destinaci do nastavení projektu. Při případné aktualizaci stačí znovu použít příkaz `pip`. Udržitelnost je tedy bezproblémová.

Přístupnost k překladům (4/5) Přístupnost k překladům je u DTM dobrá. Překlady jsou umístěné v Django admin aplikaci, která je za normálního běhu projektu spuštěna. Stačí tedy vytvořit příslušné skupiny oprávnění pro překladatele a je možné k překladům přistupovat na produkční admin aplikaci. Jelikož se překlady zálohují v databázi, není žádná zábrana v provádění překladů přímo na produkčním serveru. To znamená, že jde získat překlad od externích zdrojů jednodušeji oproti Rosetta aplikaci.

Užitečné funkcionality navíc (3/5) Cílem DTM je vylepšit pár nedostatků oproti Rosetta aplikaci. Většina změn není výrazná, jedná se například o rozšíření filtrů, či přidání možnosti vytvářet kategorie. Výraznou hodnotu

aplikaci přidává funkcionalita ukládání záloh překladů, která zpřístupňuje překlady přímo na produkčním serveru.

Cena a podmínky užití (5/5) DTM je open source řešení, které je zdarma. Dílo je dostupné pod licencí MPL 2.0.

Možnost zasahovat do řešení (4/5) Jelikož je dílo dostupné pod MPL 2.0 licencí, je možnost si celé dílo forknout a udělat v něm jakékoliv změny. Pokud by tyto změny mohly přidat hodnotu DTM aplikaci, je možnost vytvořit pull request a zažádat o zařazení úprav do originálního projektu.

Možnost revize překladu (0/5) Neposkytuje žádnou možnost označení překladů určených k revizi.

2.6.4 Lokalise

Popis řešení Lokalise je komerční a proprietární SaaS, který poskytuje lokalizační služby pro mnoho jazyků a frameworků včetně Djanga. Oproti předešlým řešením – Django basic, Rosetta a Django translation manager – poskytuje Lokalise systém řízení překladů (TMS), který přidává funkcionality zabývající se řízením a automatizací procesu lokalizace.

Celé řešení je online, na [41].

Rychlost integrace (3,5/5) Pro využití Lokalise je zapotřebí si vytvořit na službě účet a následně projekt. Poté je potřeba do projektu importovat překladové katalogy a po překladu je z něj opět exportovat. Toho lze docílit třemi způsoby. První, nejjednodušší, ale manuálně nejnáročnější je nahrávání individuálních překladových katalogů přes webové rozhraní Lokalise. Lze vybrat cestu k souborům, či jednoduše nahrát soubory pomocí drag-and-drop. Export při této variantě se provádí přes tlačítko.

Druhá varianta je nahrávat a stahovat překladové soubory pomocí Lokalise CLI nástroje. Nástroji stačí specifikovat token, projektové ID a cílové soubory. Společně s těmito základními parametry lze specifikovat více, pro uživatele individuálně, užitečných možností, např. rozpoznání podobných řetězců v různých souborech či otagování nahrávaných řetězců. V projektu tak může být vytvořen jeden i více skriptů, které budou sloužit k nahrávání a stahování překladových katalogů z Lokalise. Ukázky použití CLI jsou vidět na výpisech kódu 2.7 a 2.8.

```
lokalise --token 7e02197486aa32a83e5c10fd1992568269c1531f \
import 98305045592fb799a15d20.15846093 \
--file /myapp/locale/\*/\*.po \
--lang_iso en --tags app2.0 \
/
```

Výpis kódu 2.7: Import překladových katalogů do Lokalise.

```
lokalise --token 7e02197486aa32a83e5c10fd1992568269c1531f \
export 98305045592fb799a15d20.15846093 --type po \
/
```

Výpis kódu 2.8: Export překladových katalogů z Lokalise.

Třetí varianta je integrace Lokalise přímo s VCS repozitářem. V době psaní této práce je integrace dostupná pro GitLab, GitHub a BitBucket.

V případě GitLabu probíhá integrace následovně. Nejdříve je zapotřebí vygenerovat pro Lokalise přístupový klíč (anglicky access token) (lze vidět na obrázku 2.1). Na obrázku 2.2 je vidět nastavení Lokalise, kde se zvolí repozitář a větev ze které Lokalise bude moci importovat soubory. Následně se v Lokalise vyberou překladové katalogy, které se mají importovat. Pull (import katalogů) se dá provést manuálně, či nastavit automatickou synchronizaci. Výběr překladových katalogů a manuální pull jsou vidět na obrázcích 2.3 a 2.4.

Pokud chce uživatel překlady exportovat zpět do repozitáře, je zapotřebí projekt v Lokalise exportovat a zvolit trigger na GitLab. To zajistí vytvoření nové větve z větve, na kterou je Lokalise napojeno, vytvoření commitu se všemi upravenými soubory a vytvoření pull requestu do větve, na kterou je Lokalise napojeno. Při zvolení možnosti „Preview"se zobrazí výsledná struktura souborů a adresářů. Export lze vidět na obrázku 2.7.

Udržitelnost (4/5) Udržitelnost je závislá na způsobu integrace. Pokud je ale Lokalise integrován přímo s projektovým repozitářem, je zapotřebí pouze kontrolovat a schvalovat pull requesty.

Přístupnost k překladům (5/5) Překlady jsou dobře přístupné pro kohokoliv. S velmi dobrým a intuitivním designem se v Lokalise naučí překládat i člověk, který nemá dobré dovednosti s užíváním elektronických zařízení. Navíc jsou překlady přístupné naprosto odkudkoliv a kdykoliv.

Na obrázcích 2.6 a 2.5 je vidět projektový ovládací panel a obrazovka, na které se provádí překlady.

Užitečné funkcionality navíc (4,5/5) Lokalise obsahuje mnoho funkcionalit navíc. Obsahuje funkcionality umožňující týmovou kooperaci, správu lokalizačního workflow, automatizaci lokalizačního procesu, kontrolu kvality (QA) překladů, objednání překladů a další.

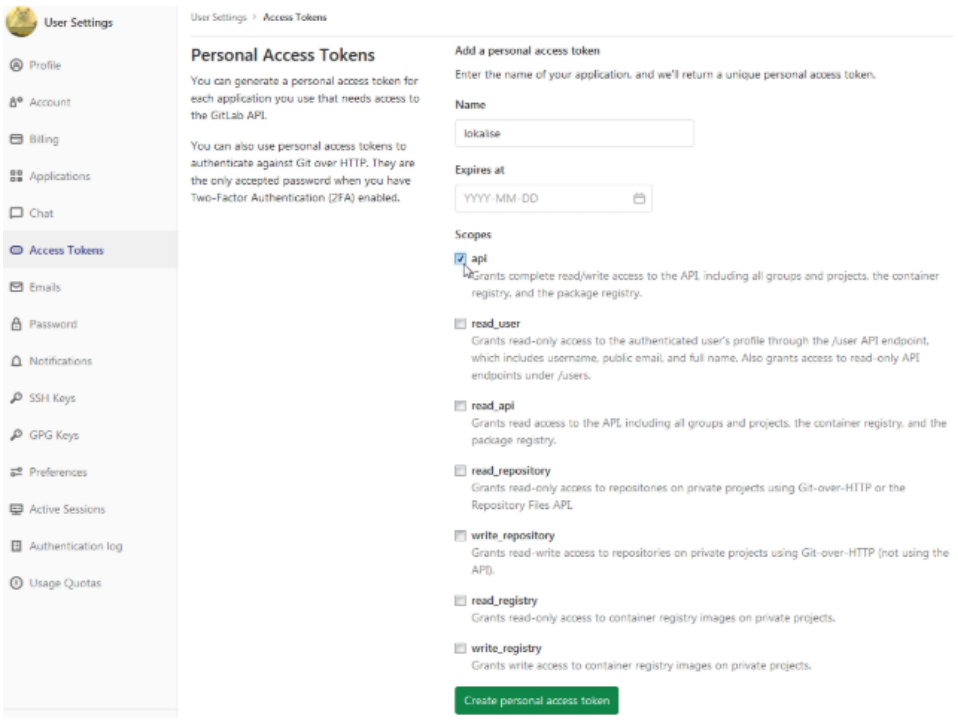
Některé funkcionality Lokalise jsou blíže rozebrány v sekci 2.7.

Cena a podmínky užití (2/5) Lokalise je komerční SaaS. Musí se za něj platit na měsíční bázi. V době psaní této studie je nejnižší cena 90 USD, nejvyšší 435 USD za měsíc. Ceník v době psaní této práce je na obrázku 2.8.

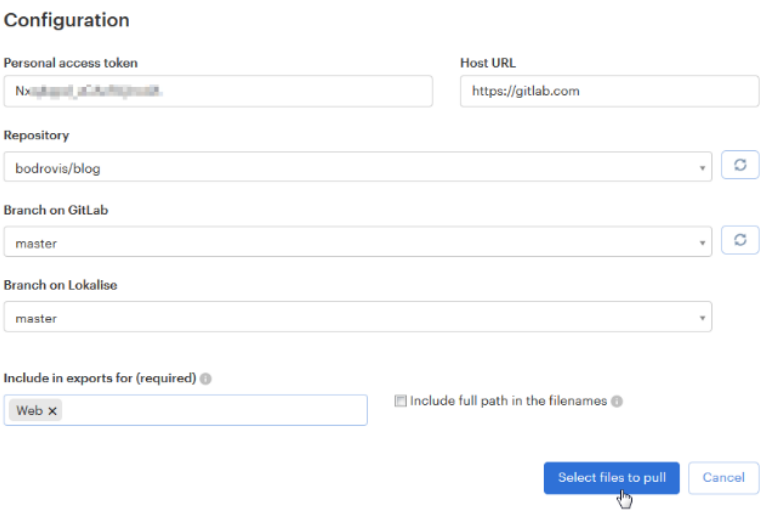
Pro open-source projekty je Lokalise poskytován zdarma.

Možnost zasahovat do řešení (0/5) Do Lokalise, jakožto do proprietárního softwaru, nelze zasahovat a vytvářet vlastní úpravy

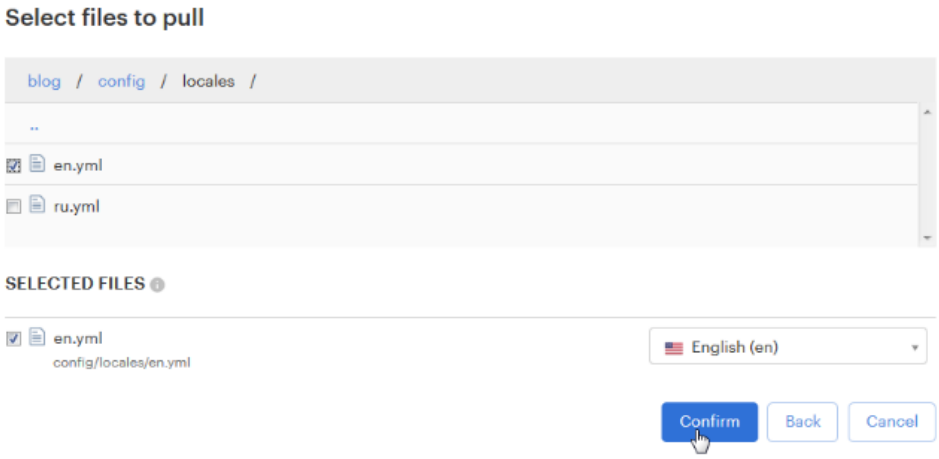
Možnost revize překladu (5/5) Lokalise podporuje revizi i více revizních cyklů pomocí zadání zřetězených úkolů (tasks). Také je možné zapnout 13 QA kontrol, například na gramatické chyby, nekonzistentní HTML tagy, odlišná data (číslo, URL, email atd.) mezi zdrojem a překladem a další.



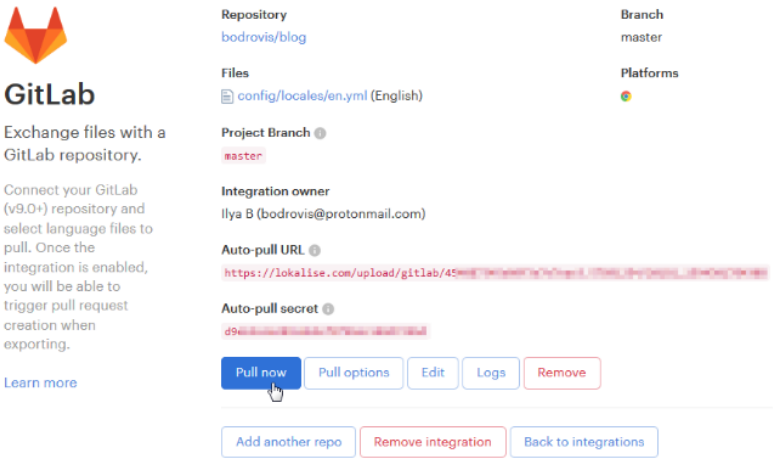
Obrázek 2.1: Vygenerování GitLab access tokenu. [42]



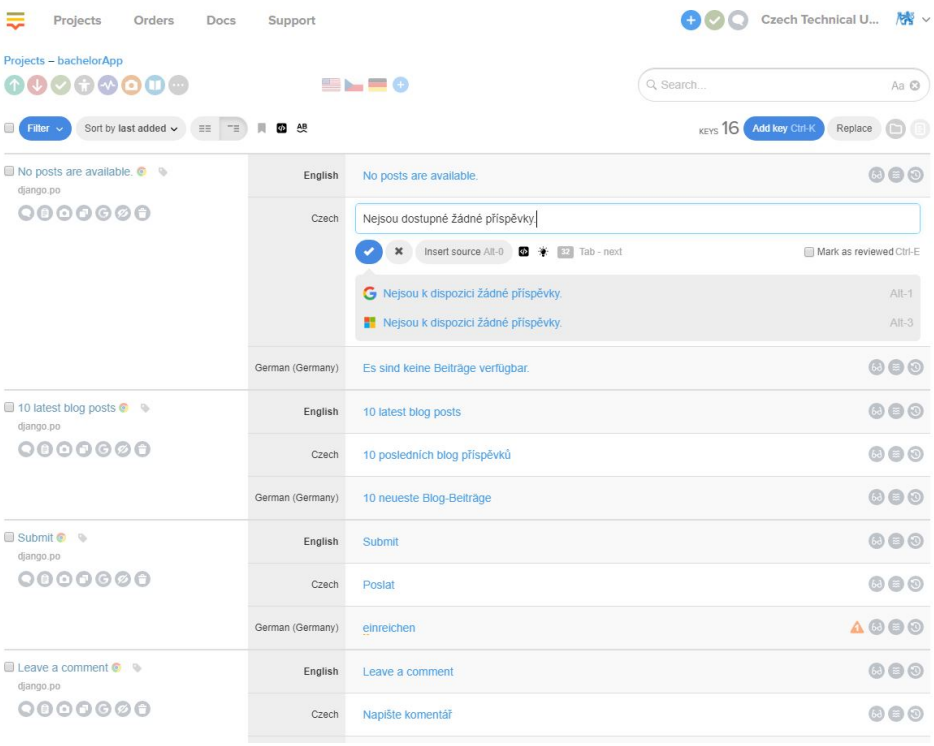
Obrázek 2.2: Lokalise konfigurace připojení ke GitLab. [42]



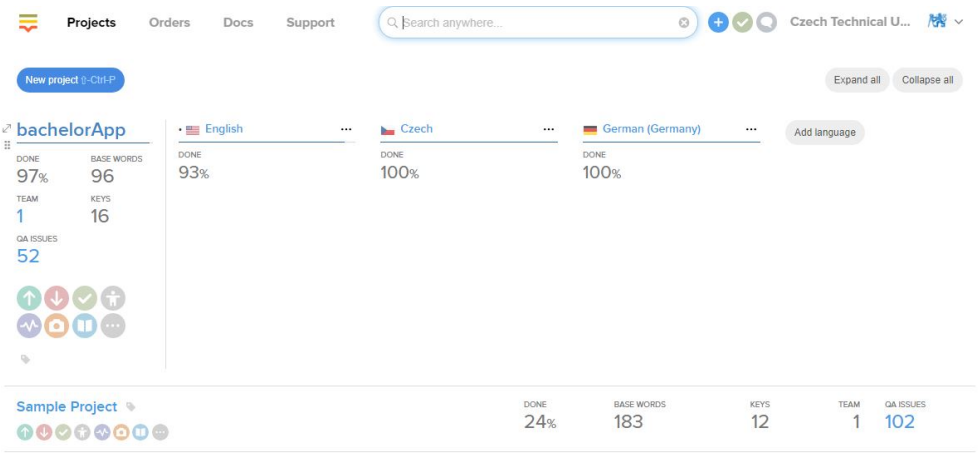
Obrázek 2.3: Výběr zdrojových souborů na importování. [42]



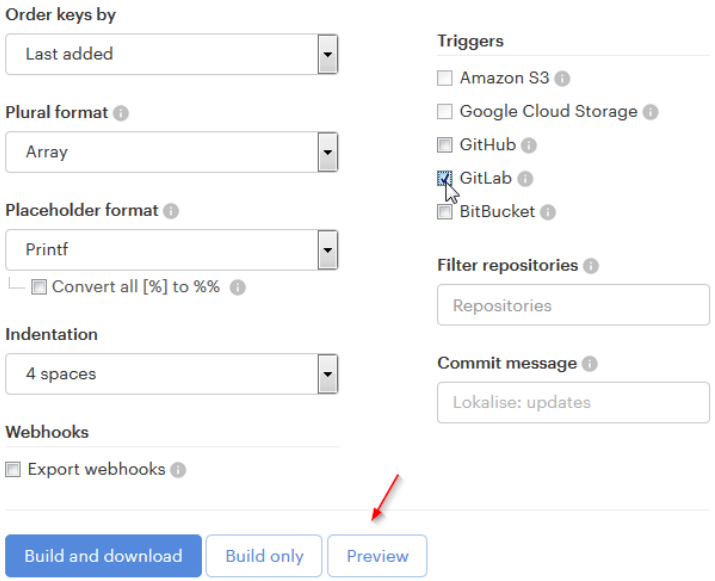
Obrázek 2.4: Manuální pull překladových katalogů. [42]



Obrázek 2.5: Lokalise obrazovka s řetězci k překladu. [41]



Obrázek 2.6: Projektový ovládací panel Lokalise. [41]



Obrázek 2.7: Export překladových katalogů do repozitáře. [42]

PLAN	Start	Essential	Pro	Enterprise
	For small teams and early stage startups	For small and medium-sized businesses	For those seeking advanced localization tools	For large businesses or those in highly regulated industries
PRICE	$90 per month with annual billing, $110/mo if billed monthly. 10 seats included, extra seats at $9/mo (or $11/mo if billed monthly)	$190 per month with annual billing, $230/mo if billed monthly. 10 seats included, extra seats at $19/mo (or $23/mo if billed monthly)	$435 per month with annual billing, $525/mo if billed monthly. 15 seats included, extra seats at $29/mo (or $35/mo if billed monthly)	To get a price estimate contact our Sales team
TOP FEATURES	✓ Unlimited projects ✓ Collaborative web-based editor ✓ Tasks ✓ Mobile SDK ✓ API and CLI tool ✓ GitHub, GitLab and BitBucket integrations ✓ Productivity integrations (Jira, Slack, Webhooks etc.) ✓ 5000 hosted keys	**Everything from Start plus** ✓ Glossary ✓ Screenshots ✓ In-context editors ✓ Translation memory ✓ Translation history ✓ Machine translations ✓ Paged documents ✓ Project activity ✓ Chained tasks ✓ Stats and reports ✓ 10000 hosted keys	**Everything from Essential plus** ✓ Branching ✓ Two-way Sketch and Figma plugins ✓ Screenshot workflows ✓ Zendesk Guides and Contentful integrations ✓ Amazon S3 and Google Cloud Storage integrations ✓ TM management ✓ Custom translation statuses ✓ User groups ✓ Shared glossaries ✓ Initial setup assistance ✓ 30000 hosted keys	**Everything from Pro plus** ✓ Dedicated manager ✓ SAML-based SSO ✓ Audit logs ✓ Feature requests ✓ Optional SLA

Obrázek 2.8: Ceník služeb Lokalise. [43]

2.6.5 Zanata

Popis řešení Zanata je webový systém určený pro překladatele k překladu dokumentace a softwaru online pomocí webového prohlížeče. Je napsán v Javě a používá moderní webové technologie jako JBoss EAP, CDI, GWT, Hibernate a REST API. V současné době podporuje překlad dokumentací DocBook a Publican prostřednictvím souborů PO a řady dalších formátů. Projekty lze nahrávat a stahovat ze serveru Zanata pomocí pluginu Maven nebo Zanata CLI [44].

Zanata je dostupná online, na [45].

Rychlost integrace (2,5/5) Pro bezplatné využití Zanata řešení je možné Zanatu spustit lokálně či na vlastním serveru. Pokud se jedná o open source projekt, je možné využít veřejnou instanci Zanaty hostovanou na `https://translate.zanata.org`. Zanata může být nainstalována za použití Dockeru nebo pomocí stažení war – web archive – souboru a nakonfigurování Jboss EAP nebo Wildfly.

Pro integraci pomocí Dockeru je zapotřebí stáhnout zanata-server repozitář dostupný online, z [46]. Následně je potřeba sestavit docker image, případně upravit konfiguraci, spustit server pomocí skriptu obsaženého v repozitáři a

vytvořit na serveru admin uživatele. Výpis kódu 2.9 ukazuje spuštění Docker serveru.

Po spuštění se dá na server přistoupit přes URL `http://HOST:PORT/zanata`, kde HOST je IP adresa serveru a PORT je defaultně nastaven na 8080. Nyní stačí ve webovém prohlížeči založit nový projekt, vytvořit jeho první překladatelskou verzi a nahrát překladové katalogy.

Zanata vyžaduje jako první nahrát soubor typu `.pot`. Tento soubor není Djangem defaultně generován, je proto zapotřebí jej vytvořit z `.po` souboru. Ve výpisu kódu 2.10 je ukázán jeden z možných postupů na vytvoření `.pot` souboru.

Pro nahrání `.pot` a `.po` souborů lze využít jeden ze tří způsobů. Drag-and-drop přímo ve webovém rozhraní, Maven plugin nbeo Zanata CLI.

Pro nahrání souborů pomocí Zanata CLI se musí nainstalovat Zanata CLI pomocí 0install. Instalace je názorně ukázána na výpisu kódu 2.11.

Soubory se dají nahrávat a stahovat pomocí příkazů `zanata-cli push` a `zanata-cli pull`. Užití příkazů je ukázáno ve výpisu kódu 2.12. Pro zrychlení procesu nahrávání a stahování je vhodné vytvořit soubor `zanata.xml`, ve kterém se nastaví projektová konfigurace, uživatelská (autorizační) konfigurace a nastavení lokace překladových katalogů a `.pot` souborů.

Eventuelně je možné vytvořit skripty manipulující se Zanata CLI, které budou spuštěny vždy při importu a exportu překladových souborů.

```
docker build -t zanata/server .
./runapp.sh
docker cp conf/admin-user-setup.sql zanatadb:/tmp
docker exec -it zanatadb bash -c "mysql -u zanata \
--password=password zanata < /tmp/admin-user-setup.sql" \
/
```

Výpis kódu 2.9: Instalace a spuštění Zanata serveru

```
msgfilter -i locale/en/LC_MESSAGES/django.po -o locale/source.pot true
```

Výpis kódu 2.10: Vytvoření `.pot` souboru.

```
sudo apt-get install zeroinstall-injector openjdk-8-jre
mkdir -p ~/bin
0install destroy zanata-cli
0install -c add zanata-cli\
https://raw.githubusercontent.com/zanata/\
zanata.github.io/master/files/0install/zanata-cli.xml\
/
0install -c update zanata-cli
```

Výpis kódu 2.11: Instalace Zanata CLI.

```
zanata-cli push --push-type trans --trans-dir trans
zanata-cli pull --src-dir src --trans-dir trans --create-skeletons
```

Výpis kódu 2.12: Ukázka práce se Zanata CLI.

Udržitelnost (3,5/5) Udržitelnost je závislá na způsobu integrace. Pokud projekt nemá integrovaný Maven plugin nebo Zanata CLI, je udržitelnost z důvodu náročné manipulace s překladovými katalogy velmi špatná.

Pokud je ovšem nahrávání souborů zajištěno pomocí Maven pluginu, či Zanata CLI, pak je udržitelnost lepší. Je zapotřebí pouze měnit konfiguraci těchto nástrojů, když se změní nebo přidá lokace překladových katalogů.

Přístupnost k překladům (5/5) Překlady jsou na vlastním serveru a jsou tedy přístupné online pro kohokoliv, komu byl udělen přístup. Dle autorova zkoumání působí webové rozhraní Zanaty jednoduše, moderně a má intuitivní design, ve kterém je po krátkém seznámení jednoduché se vyznat. Na obrázcích 2.9, 2.10, 2.11 a jsou ukázky rozložení a designu Zanaty.

Užitečné funkcionality navíc (3,5/5) Hlavní funkcionality Zanaty umožňují týmovou kooperaci, správu lokalizačního workflow, automatickou kontrolu kvality (QA) překladů, vytváření účtů se systémem rolí a skupin, verzování překladů, znovupoužití již přeložených frází a glosář pro projekt specifických pojmů.

Některé funkcionality Zanaty jsou blíže rozebrány v kapitole 2.7.

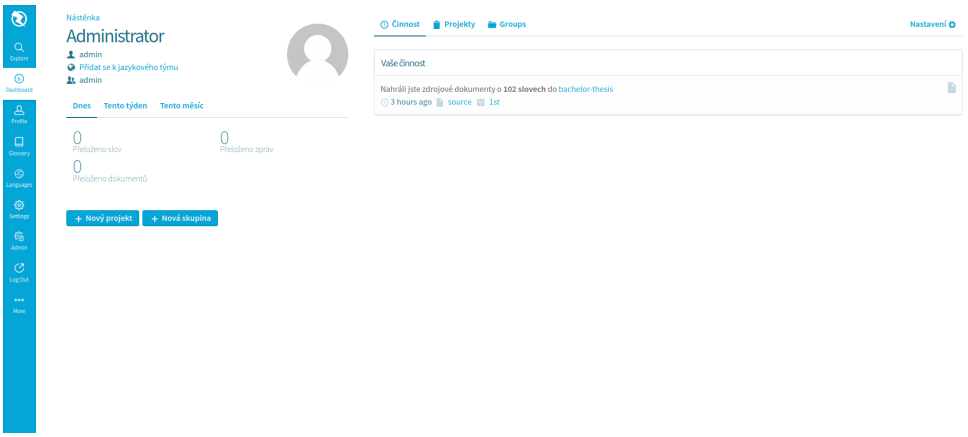
Cena a podmínky užití (5/5) Zanata je zdarma. Dílo je dostupné pod GNU Lesser General Public licencí verze 2.1.

Možnost zasahovat do řešení (4,5/5) Jelikož je dílo dostupné pod GNU Lesser 2.1 licencí, je možnost si celé dílo forknout a udělat v něm jakékoliv změny, pokud by ale uživatel takové řešení chtěl dále distribuovat, bude ho muset poskytnout pod stejnou (nebo kompatibilní) open source licencí. Pokud

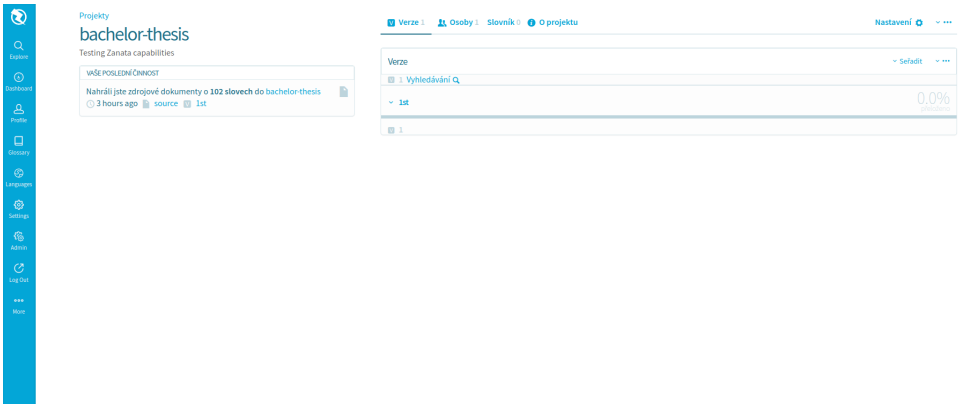
by tyto změny mohly přidat hodnotu celému projektu, je možnost vytvořit pull request a zažádat o zařazení úprav do originálního projektu.

Oproti již hodnoceným open source řešením je u Zanaty možnost před pushem či pullem překladových souborů – pomocí Zanata CLI nebo Maven pluginu – použít příkazové hooky. Tato funkcionalita velmi zlepší možnost vlastního zasahování do řešení bez zásahu do zdrojového kódu projektu samotného. Nebude tedy zapotřebí vydávat úsilí na udržování forknutého projektu.

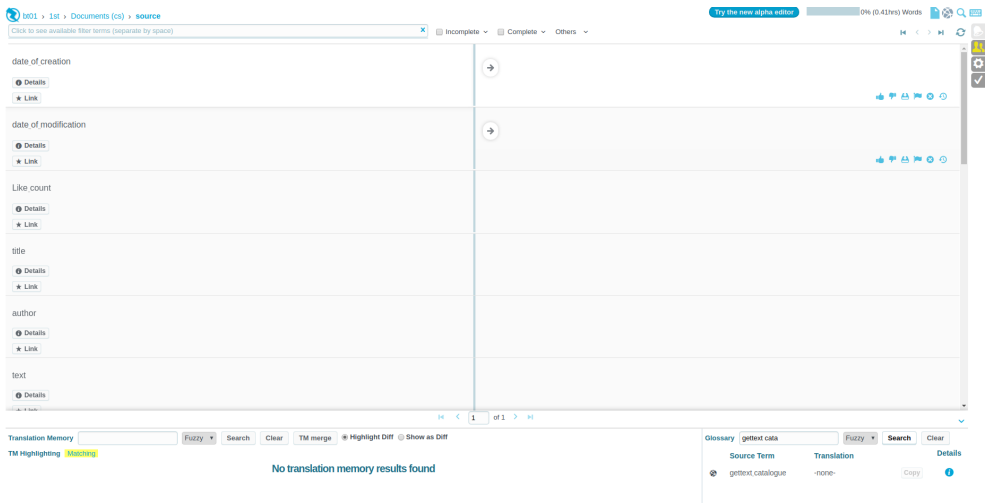
Možnost revize překladu (4/5) V Zanata projektu je speciální role reviewer, která se přiděluje uživateli pro specifický jazyk. Uživatel poté může schvalovat a zamítat hotové překlady do jazyka, u kterého má revize povoleny. Při zamítnutí je nutné napsat důvod zamítnutí překladu. V nastavení projektu je také možné nastavit QA kontroly.



Obrázek 2.9: Základní přehled v Zanata aplikaci.



Obrázek 2.10: Zanata projektový přehled.



Obrázek 2.11: Zanata editor překladů.

2.6.6 Weblate

Popis řešení Weblate je projekt napsaný v Django frameworku, který poskytuje systém pro řízení překladů ve webovém rozhraní. Je distribuován pod open source licencí a poskytuje za poplatek hosting serveru a podporu.

Weblate je dostupný online, na [47].

Rychlost integrace (2,5/5) Při zvolení možnosti využití Weblate projektu s vlastním hostingem je zapotřebí podobně jako u Zanaty spustit vlastní server. Instalace je prakticky dostupná pomocí Dockeru. Jako první krok je

potřeba stáhnout repozitář s docker-compose a vytvořit v něm konfigurační soubor `docker-compose.override.yml`, který umožňuje nastavení proměnných prostředí. Stažení repozitáře a konfigurační soubor jsou ukázány na výpisech kódu 2.13 a 2.14. Poté by pomocí `docker-compose up` mělo být možné spustit Weblate Docker kontejnery.

Weblate server běží defaultně na adrese `http://localhost:80`. Ve webovém rozhraní je možné založit i do něj vložit jednotlivé překladové katalogy (podle Weblate názvosloví *komponenty*).

Pro nahrání překladových katalogů lze využít nahrávání pomocí drag-and-drop přímo ve webovém prohlížeči nebo integrace s Git repozitářem či wlc (Weblate CLI klient). Pro správu projektu – včetně nahrávání a stahování souborů – pomocí wlc je zapotřebí stáhnout wlc přes příkaz `pip install wlc`. Ukázka konfigurace wlc v souboru `/.config/weblate` lze vidět na výpisu kódu 2.15.

Pro integraci s Git repozitářem se nejčastěji používá metoda založená na SSH. Nejdříve je zapotřebí v cílovém repozitáři přidat mezi povolené SSH klíče Weblate public SSH klíč, který je dohledatelný ve webovém rozhraní Weblate. Poté je možné do systému přidat překladové katalogy. Stačí jednoduše vejít do vytvořeného projektu a kliknout na tlačítko „přidat komponentu." Na obrázku 2.12 je formulář, který se objeví.

Po vyplnění formuláře lze zvolit možnost nahrát z verzovacího nástroje. Weblate automaticky projde repozitář a najde překladatelné zdroje, ze kterých si uživatel vybere. Ukázka výběru je na obrázku 2.13.

Nyní, když jsou ve Weblatu nahrané komponenty, má uživatel možnost nastavit automatické aktualizování překladových katalogů, či manuálně aktualizovat překlady pomocí skriptu. Ukázka takového skriptu je vidět ve výpisu kódu 2.16. Ve webovém rozhraní, v projektovém přehledu, je možnost „Spravovat."Po kliknutí na tuto možnost je uživatel odkázán na obrazovku zobrazenou na obrázku 2.14. Na této obrazovce je možné odesílat a přijímat změny ze vzdáleného repozitáře.

```
git clone https://github.com/WeblateOrg/docker-compose.git \
weblate-docker \
/
cd weblate-docker
```

Výpis kódu 2.13: Stáhnutí Weblate docker repozitáře.

```
version: '3'
services:
  weblate:
    ports:
      - 80:8080
    environment:
      WEBLATE_ALLOWED_HOSTS: weblate.test.com,localhost
      WEBLATE_ADMIN_PASSWORD: admin
      WEBLATE_ADMIN_EMAIL: admin@weblate.com
```

Výpis kódu 2.14: Ukázka konfiguračního souboru Weblate docker serveru.

```
[weblate]
url = https://hosted.weblate.org/api/

[keys]
https://hosted.weblate.org/api/ = APIKEY
```

Výpis kódu 2.15: Konfigurační soubor wlc klienta.

```
# Lock Weblate translation
wlc lock
# Push changes from Weblate to upstream repository
wlc push
# Pull changes from upstream repository to your local copy
git pull
# Update translation files, this example is for Django
./manage.py makemessages --keep-pot -a
git commit -m 'Locale updates' -- locale
# Push changes to upstream repository
git push
# Tell Weblate to pull changes (not needed if Weblate
# follows your repo automatically)
wlc pull
# Unlock translations
wlc unlock
```

Výpis kódu 2.16: Ukázka skriptu pro manuální aktualizaci překladových katalogů na Weblate. [48]

Udržitelnost (4/5) Udržitelnost je závislá na způsobu integrace. Pokud je Weblate integrován s projektovým repozitářem a je zvoleno automatické pushování změn, pak je zapotřebí pouze kontrolovat a schvalovat pull requesty.

Přístupnost k překladům (5/5) Weblate má moderní intuitivní design, ve kterém je jednoduché se orientovat. Překlady jsou online a dá se k nim přistupovat přes webový prohlížeč. Ukázky obrazovek z Weblate jsou na obrázcích 2.15, 2.16 a 2.17.

Užitečné funkcionality navíc (4,5/5) Weblate obsahuje mnoho funkcionalit. Pro příklad zde je vyčteno pouze pár významnějších. Některé z funkcionalit jsou blíže rozebrány v sekci 2.7.

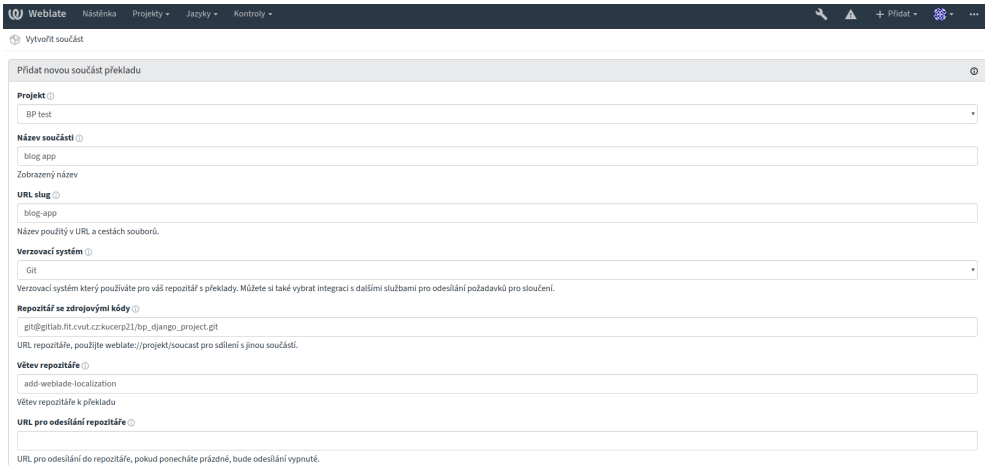
Weblate obsahuje funkcionality, které například umožňují automatizovaný lokalizační proces, týmovou kooperaci, verzování překladů (i napříč různými větvemi repozitáře), glosáře, reporting pro řízení procesu a úpravu funkčnosti Weblate pomocí doplňků.

Cena a podmínky užití (5/5) Weblate je zdarma. Dílo je dostupné pod GNU GPL verze 3+.

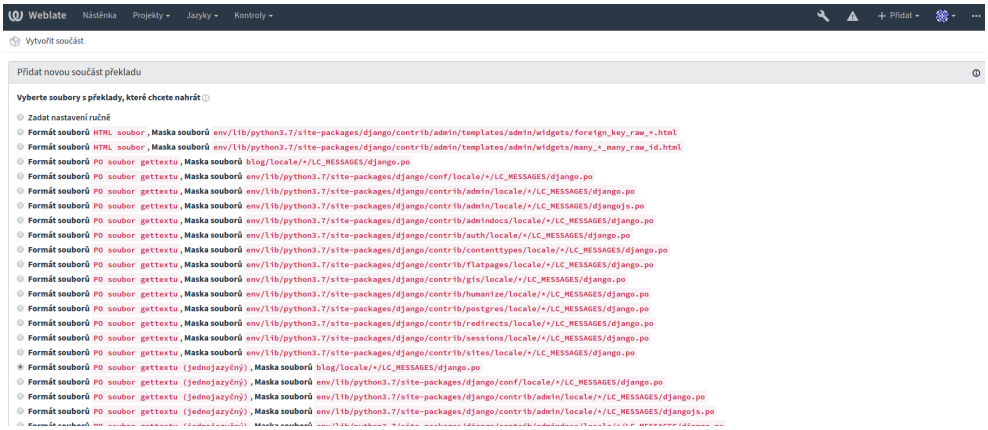
Možnost zasahovat do řešení (5/5) Jelikož je dílo dostupné pod GNU GPL v.3+ licencí, je možnost si celé dílo forknout a udělat v něm jakékoliv změny, pokud by ale uživatel takové řešení chtěl dále distribuovat, bude ho muset poskytnout pod stejnou (nebo kompatibilní) open source licencí. Pokud by tyto změny mohly přidat hodnotu celému projektu, je možnost vytvořit pull request a zažádat o zařazení úprav do originálního projektu.

Co ovšem dílo opravdu dělá upravitelné dle vlastních požadavků, je možnost vytvářet vlastní doplňky (addons). Doplňky poskytují způsob, jak přizpůsobit proces překládání. Mohou být nainstalovány v zobrazení překladových komponenty a mohou pracovat na pozadí aplikace [49]. Více o doplňcích v sekci 2.7.

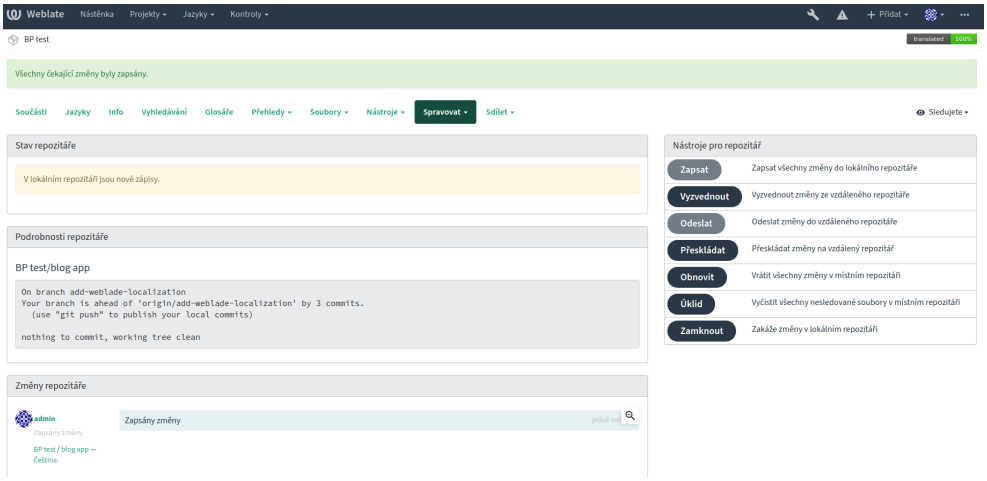
Možnost revize překladu (4,5/5) Weblate obsahuje nastavení workflow, kde je možné si vybrat jeden z více způsobů, jak bude probíhat lokalizační proces. Způsoby revizí se liší především v tom, kdo je může provádět. Při zvolení způsobu Peer reviews může kdokoliv k překladu přidat návrh, o kterém je následně hlasováno a může být přijat. Druhý, více známý, způsob je zvolení možnosti Dedicated reviewers. Tento způsob je obvyklejší u větších projektů, které mají dedikované osoby provádějící revize.



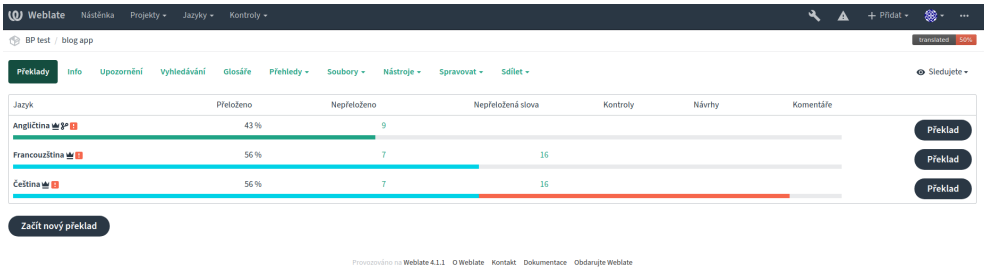
Obrázek 2.12: Přidání nové komponenty do Weblate apliakce.



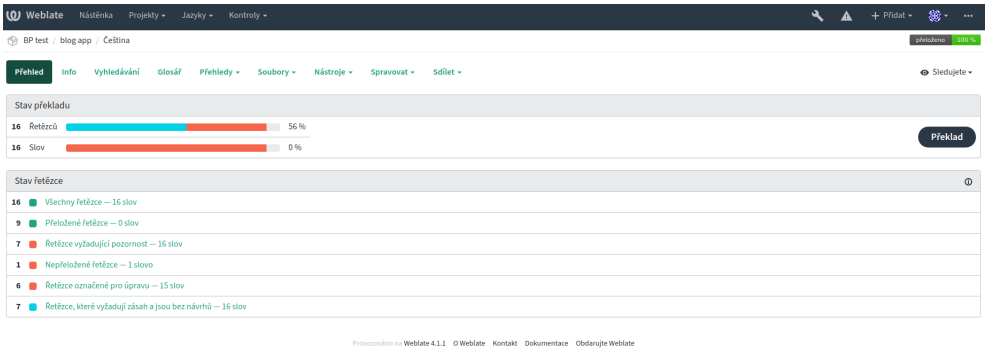
Obrázek 2.13: Weblate výběr překladových katalogů, podle nalezených možností v repozitáři.



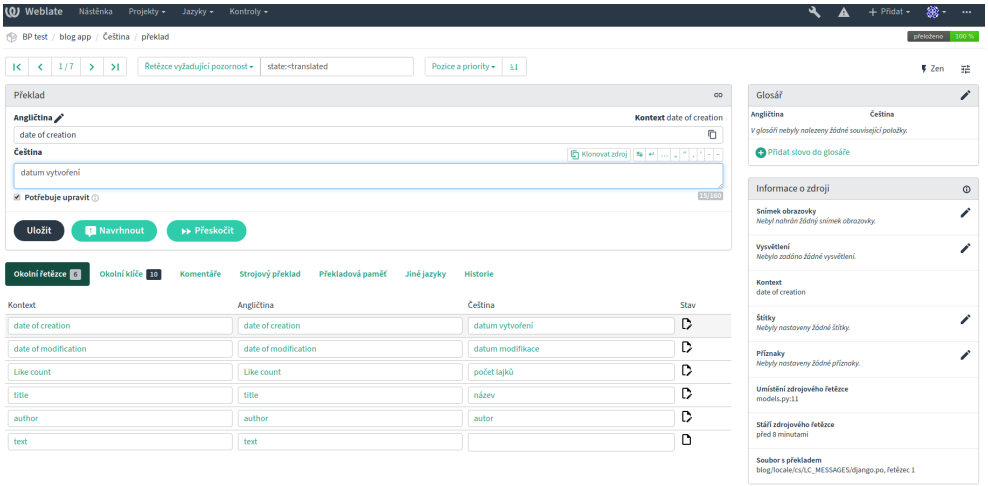
Obrázek 2.14: Synchronizace změn s repozitářem ve Weblate.



Obrázek 2.15: Weblate projektový přehled.



Obrázek 2.16: Weblate zobrazení přehledu jazyka.



Obrázek 2.17: Weblate editor překladových souborů.

2.7 Rozbor užitečných funkcionalit

V této sekci jsou rozebrány užitečné funkcionality, které mají některá z řešení implementována. Je zde nastíněno, jak fungují a co přesně poskytují.

Po krátké diskuzi autora bakalářské práce s Vladislavs Andre – Lokalise Inbound Sales Representative – a Lev Duman – Lokalise Sales Executive – bylo zjištěno, jaké funkcionality jsou v Lokalise považovány za nejdůležitější pro uživatele.

Vysoká podpora týmové práce V Lokalise je podpora týmové práce považována za jednu z nejhlavnějších priorit. Podpora lepší týmové práce stojí jak v Lokalise, tak v jiných aplikacích na různých funkcionalitách. Například se jedná o přehledné monitorovací statistiky pro manažery, možnost vytvářet různé týmové role, možnost nastavení specifického workflow, či funkcionalita umožňující zadávání a plnění úkolů (anglicky tasks). Lokalise například poskytuje i integraci do trackovacích systému jako JIRA, Trello a ASANA.

Možnost zobrazit si strojový překlad Strojový překlad se dá použít pouze jako rada, ale může být zároveň použit i jako hlavní překlad pro testovací fázi či jako dočasný zástupný text.

V Lokalise se strojový překlad řeší přes API. Při využití strojového překladu se podle dostupnosti překladů do daného jazyka zobrazí překladately návrh pomocí překladačů od firem Google, Microsoft a Amazon. Každá z firem poskytuje své vlastní API pro překládání.

Objednání překladů V Lokalise je možné si objednat profesionální překlady. Dají se ale objednat pouze přímo od Lokalise nebo jedné další firmy, Gengo.

Integrace s kódovým repozitářem Tato funkcionalita byla také označena zaměstnanci Lokalise jako jedna z nejdůležitějších a uživateli nejvyužívanějších. Přináší hlavně výhodu v lepší udržitelnosti integrace překládacího systému, jelikož se můžou překlady automaticky aktualizovat.

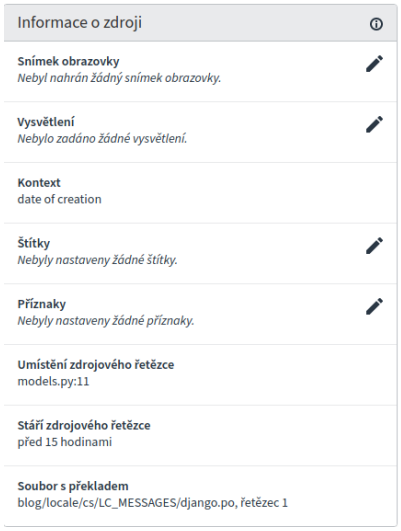
Umožňuje to automatizovat celý proces lokalizace. Workflow lokalizace je také možné přizpůsobit, kvůli možnosti nastavení různých cílových větví pro překlady.

Integrace s repozitářem je implementována v řešeních Lokalise a Weblate. U obou řešení funguje na základě SSH protokolu a je dostupná pouze pro verzovací systém Git.

Přidání kontextu k překladům Další, dle Lokalise týmu, důležitou funkcionalitou je možnost přidávání kontextu k překladům. K vytvoření dobrého překladu je zapotřebí znát dost okolností. Je zapotřebí například vědět, kde je text umístěn, co je přesně textem myšleno a zda je text limitovaný na nějaký počet charakterů.

Defaultně je v gettext možnost přidat textový kontext, který všechny požadavky na vytvoření správného překladu nemůže samostatně splnit.

Zanata i Weblate obsahují více možností jak k překladu přidat kontext. Kontext k překladu je možné přidat pomocí snímku obrazovky, videa, komentářů, štítků, či příznaků. Jak Lokalise, tak Weblate zobrazují u překladu v jakém souboru a kde se nachází. Ukázka kontextového menu Weblate je vidět na obrázku 2.18.



Obrázek 2.18: Kontextového menu ve Weblate.

Znovupoužití již přeložených frází Zanata má funkcionalitu, díky které se dynamicky vyhledávají podobné, již přeložené, řetězce napříč projektem. Překladateli se tak zobrazují při překladu podobné překlady. Díky tomu se lépe udržuje konzistentní překládání klíčových slov a sousloví.

Glosář pojmů Glosář je funkcionalita, která je obsažena v Lokalise, Zanata i Weblate projektech. Vztahuje se na celý překladatelský projekt. Jedná se o slovník, do kterého by se měla napsat všechna pro projekt klíčová, specifická, či vymyšlená slova. Ve Weblate se například glosář zobrazuje překladatelům vždy při překládání. Pokud se ve zdrojovém řetězci vyskytuje slovo přítomné v glosáři, zobrazí se překladateli překlad pro toto slovo. Ukázka na obrázku 2.19.

Obrázek 2.19: Weblate glosář.

Kontroly zajištění kvality Kontroly zajištění kvality, neboli QA kontroly, jsou implementovány v Lokalise a Weblate. V obou projektech je více kontrol, které lze zapnout dle uživatelovy preference, či projektových požadavků. Jde

například o kontrolu stejných čísel, URL, e-mailů v přeloženém řetězci oproti zdrojovému řetězci, kontrolu přebývajících bílých znaků a další.

Jedná se o kontroly, které je možné provést počítačově. Kontroly jsou prováděny automaticky systémem. Pokud pro nějaký překlad kontrola nevyjde, pro daný překlad se uživateli zobrazí upozornění. Ve Weblate je možné si vybrat u každé kontroly, jestli bude zobrazovat pouze upozornění, či chybu.

Příkazové hooky Zanata poskytuje možnost nastavení příkazových hooků. Spouštějí se podle nastavení Zanaty před, nebo po specifikovaných Zanata CLI nebo Maven plugin příkazech. Spouštěný příkaz může být téměř jakýkoliv příkaz, který je spustitelný v příkazové řádce.

Dá se tak docílit lehkého přizpůsobení – spíše přidání funkcionality, než upravení – Zanaty bez nutnosti zasahovat do zdrojového kódu.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<config xmlns="http://zanata.org/namespace/config/">

...

  <hooks>
    <hook command="push">
        <before>
        po4a-gettextize -f man -m manpage.1 -p manpage.pot
        </before>
        <after>rm -f manpage.pot</after>
    </hook>
    <hook command="pull">
        <after>
        po4a-translate -f man -m manpage.1 -p trans/de/manpage.po
        -l manpage.de.1 --keep 1
        </after>
        <after>rm -rf trans</after>
    </hook>
  </hooks>

...

</config>
```

Výpis kódu 2.17: Ukázka nastavení hooků v souboru `zanata.xml`. [50]

Možnost zasahování do aplikace pomocí doplňků Weblate poskytuje možnost upravit překladový workflow pomocí doplňků (anglicky addons). Ve

Weblate jsou defaultně vestavěné doplňky umožňující například automatický strojový překlad, upravování `.po` souborů, aby byly kompatibilní s `.pot`, squashování git commitů a další.

Existuje ale možnost vytvořit vlastní doplňky, pomocí kterých se dá upravit funkcionalita Weblate do značné míry, aniž by se muselo zasahovat do zdrojového kódu. Programují se v Django frameworku, na němž je celý Weblate postavený. Vytvoří se pomocí přetížení třídy `BaseAddon` a určí se, při jakých událostech (anglicky events) by měl být doplněk notifikován.

Funkcionalita je tedy do nějaké míry podobná hookům, ale je lépe integrovaná a je možno s ní dosáhnout většího přizpůsobení aplikace.

Verzování překladů Překlady v systému Zanata jsou postaveny na systému verzí. Všechny překladové soubory se musí vázat k nějaké verzi překladového projektu. Všechny překlady se tedy dají verzovat a v aplikaci je možnost mít odlišné verze překladů pro odlišné verze aplikace. Zároveň slouží starší verze jako záloha. Pro případné procesy, jako je rebranding projektu, mohou být verze také velmi užitečné.

Podpora větví Weblate má namísto verzování překladů funkcionalitu, která dovoluje mít různé verze překladů, které se vážou na rúzné větvě Git repozitáře. Výhody této funkcionality jsou prakticky stejné jako u 2.7. Velké pozitivum podpory větví je, že je přímo navázáno na strukturu repozitáře a celkově je tedy snadnější synchronizace překladů s jednotlivými větvemi.

Zadávání úkolů Lokalise poskytuje správcům možnost zadávat překladatelům úkoly a sledovat jejich proces.

Úkoly je možné zřetězit. Například ihned po přeložení je možné nastavit zadání úkolu revize na daném překladu.

U úkolů se dá nastavit například datum uzávěrky, typ úkolu (revize nebo překlad), rozsah a řešitel. Z úkolu se dá do budoucna vytvořit šablona, pomocí které se dá jednoduše zadat úkol znovu.

2.8 Výstup analýzy

Po důkladné analýze vybraných řešení a vyhodnocení jednotlivých aspektů bylo vybráno jako nejlepší existující řešení Weblate. Shrnutí hodnocení všech řešení je v tabulce 2.1. Řešení je open source a dá se používat zdarma, pokud je uživatel ochoten hostovat vlastní server. Zároveň má nejvíce funkcionalit, které řešení značně přidávají hodnotu. Udržitelnost je přitom velmi nenáročná za cenu náročnější integrace. Řešení Weblate je ze všech nejlépe individuálně upravitelné pomocí funkcionality doplňků. Dle GitHub statistik je ze všech open source řešení nejvíce forkované, má nejvíce kontributorů, nejvíce přidělených hvězd a nejvíce schválených pull requestů, za poslední měsíc [51].

Řešení, které neběží jako samostatný projekt – Django basic, Rosetta a Django translation manager – mají vynikající rychlost integrace a udržitelnost. Zdaleka ale neobsahují tolik funkcionalit jako ostatní analyzovaná řešení.

Zanata je dobré řešení, které je open source a dá se používat zdarma. Weblate ovšem oproti Zanatě obsahuje funkcionality, které mu přidávají značnou hodnotu, například doplňky, integrace s repozitářem, rozsáhlejší podpora přidávání kontextu, či kontroly zajištění kvality.

Lokalise je placené řešení, což mu velmi snižuje hodnocení. Z hlediska obsažených funkcionalit je na tom stejně jako Weblate. Většinu důležitých funkcionalit obsahují obě řešení. Významné funkcionality, které Weblate postrádá, jsou možnost objednání profesionálních překladů a rozšířená podpora projektové správy.

Aspekt:	**Django basic**	**Rosetta**	**DTM**	**Lokalise**	**Zanata**	**Weblate**
Rychlost integrace	5	4	4	3,5	2,5	2,5
Udržitelnost	5	5	4,5	4	3,5	4
Přístupnost k překladům	0	2	4	5	5	5
Užitečné funkcionality navíc	0	1	3	4,5	4	4,5
Cena a podmínky užití	5	5	5	2	5	5
Možnost zasahovat do řešení	4	4	4	0	4,5	5
Možnost revize překladu	0	0	0	5	4	4,5

Tabulka 2.1: Shrnutí hodnocení jednotlivých řešení. Maximální hodnocení jednoho kritéria je 5.

2.9 Diskuze analýzy existujících řešení

2.9.1 Určení dalšího postupu studie

Dle výsledků analýzy existujících řešení je patrné, že nejlepší dosavadní existující řešení je Weblate. Tento fakt je více odůvodněn v sekci 2.8.

Jelikož je řešení Weblate velmi obsáhlé, kvalitní, dostatečně hodnotné, open source a jeho užívání je zdarma, bude se v této práci řešit vylepšení tohoto řešení místo vytváření řešení nového.

2.9.2 Významné chybějící funkcionality Weblate

Weblate je sice řešení, které obsahuje stejně jako s Lokalise ze všech řešení nejvíce funkcionalit, ale stále jsou funkcionality, které postrádá a přitom jsou v jiných řešeních obsažena.

První je možnost získání překladu od externích zdrojů přímo v aplikaci. Díky této funkcionalitě by bylo možné objednat si překlad od individuálních překladatelů na volné noze či firem orientovaných na překlad. Bylo by tak

docíleno rychlejšího získání překladatelů a jednodušší distribuce překladových řetězců přímo k překladatelům.

Druhá varianta je přidání funkcí pro rozšíření možností projektové správy. Tím je myšleno například přidělování úkolů překladatelům, plánování vydání, reporting odvedené práce, časové odhady atd. Pomocí těchto funkcionalit by mohl být proces lokalizace celkově zefektivněn a zrychlen, obzvláště pro větší týmy. Weblate by s rozšířenou podporou projektové správy poskytoval spektrum funkcionalit, díky kterým by pro projektové (lokalizační) správce bylo jednodušší řízení lokalizace.

2.9.3 Výběr funkcionality pro zkoumání

Pro návrh bylo vybráno rozšíření podpory projektového managementu. Funkcionalita byla vybrána s ohledem na elektronickou korespondenci mezi autorem práce a Ing. Michalem Čihařem – zakladatelem Weblate – o chybějících funkcionalitách a následném projednání s vedoucím práce.

Kapitola 3

Konceptuální návrh

3.1 Funkcionality projektové správy chybějící ve Weblate

Pro efektivní projektový management jsou nejdůležitější schopnosti:

- rozvrhovat a plánovat,
- dobře kooperovat v rámci týmu,
- monitorování času,
- reporting postupu a
- rozvrhování rozpočtu projektu [52, 53].

Pro docílení co nejlepší pomoci s některými aspekty projektové správy je možné implementovat funkcionality, které řadu z nich pokryjí. Jednotlivé funkcionality, které Weblate postrádá a značně by pomohly projektové správě, jsou popsány v této sekci.

3.1.1 Vytváření úkolů

Všechny funkcionality, které přinesou projektovým správcům značnou kontrolu nad projektem a jeho plánováním, stojí na možnosti zadávání úkolů. Bude přidána možnost vytvářet úkoly, které se vztahují přímo na překladové katalogy. Samotné zadávání úkolů má vysokou přidanou hodnotu v plánování projektu. Jde na něm ale vytvořit více funkcionalit, které velmi pomůžou s projektovou správou.

Tato funkcionalita samotná u sebe bude evidovat

- název,
- popis úkolu,

- řešitele,
- soubory nebo klíče týkající se úkolu,
- prioritu a
- konečný termín.

Další parametry, které by se daly evidovat, jsou považovány za rozšíření jiných funkcionalit.

3.1.2 Úkoly s prerekvizitami

Pomocí této funkcionality bude možné úkoly řetězit a vytvářet jim tak prerekvizity. Bude tedy patrné, co je potřeba přeložit, či zkontrolovat dříve. Díky tomu bude možné naplánovat více úkolů dopředu. Prakticky to znamená přidání atributu prerekvizita při zakládání nového úkolu, kde půjdou vybrat úkoly, na kterých nový úkol závisí.

3.1.3 Vytváření týmů

Schopnost vytvářet týmy nám umožní vytvářet skupiny překladatelů pro různé jazyky, případně překladatelů kvalifikovaných na revize atd. K úkolům tedy bude možno přiřadit jako řešitele celý tým.

3.1.4 Plánování milníků

Tato funkcionalita umožní vytvoření milníku. Jednotlivé úkoly půjde přiřadit pod milník. Pomocí časových odhadů, zřetězení úkolů, priority a časového vytížení jednotlivých překladatelů pak bude možné vypočítat, za jak dlouho bude milníku dosaženo. Milníků bude možné mít vytvořeno více najednou a budou moci být plněny paralelně. U milníku bude evidováno:

- název,
- popis a
- úkoly spadající do milníku

3.1.5 Časové odhady

Jedná se o rozšíření úkolů, které umožní ke každému úkolu přidat časový odhad, jak dlouho bude trvat jeho dokončení. Toho se dá následně využít pro jiné funkcionality.

Časový odhad úkolu bude možné odhadnout aplikací, která vezme v potaz přiřazené komponenty a v závislosti na počtu klíčů nastaví podle určeného koeficientu časový odhad. Odhad bude poměrně nepřesný z toho důvodu, že se

úkol může týkat pouze pár klíčů, či naopak překladu komponenty do více jazyků, pro rychlé nastavení časového odhadu určeného k odhadu dokončení milníku bude ale dostačující.

Časové odhady milníků budou vytvořeny pomocí Ganttova grafu. Prerekvizita pro vytvoření časového odhadu milníku je nutnost, aby všechny úkoly obsahovaly časový odhad. Ganttův diagram bude sestaven pomocí časových odhadů jednotlivých úkolů, řešitelů úkolů, prerekvizitních úkolů a priority úkolů. Pokud nějaké úkoly nebudou mít nastavené řešitele, bude se v Ganttově diagramu předpokládat, že mají všechny řešitele stejného.

3.1.6 Monitoring milníků

Pomocí monitoringu milníků bude projektový správce schopen sledovat stav milníků a zjišťovat tak informace potřebné k úspěšnému řízení lokalizace. Monitorovací informace o milníku bude zobrazena při zobrazení detailu milníku.

3.2 Případy užití

Konceptuální návrh, jakožto technická část studie proveditelnosti, bude realizován pouze v rámci definování případů užití. Základní případy užití budou definovány pro všechny určené funkcionality podporující projektové řízení. Na obrázku 3.1 je diagram případů užití.

UC1: Vytvoření úkolu

Umožňuje uživateli s oprávněním projektový manažer vytvořit úkol s různými parametry. Uživatel povinně zadá název, popis úkolu. Volitelně pak může zadat řešitele (i tým), soubory týkající se úkolu, konečný termín, prerekvizitní úkoly, prioritu, časový odhad a příslušné milníky.

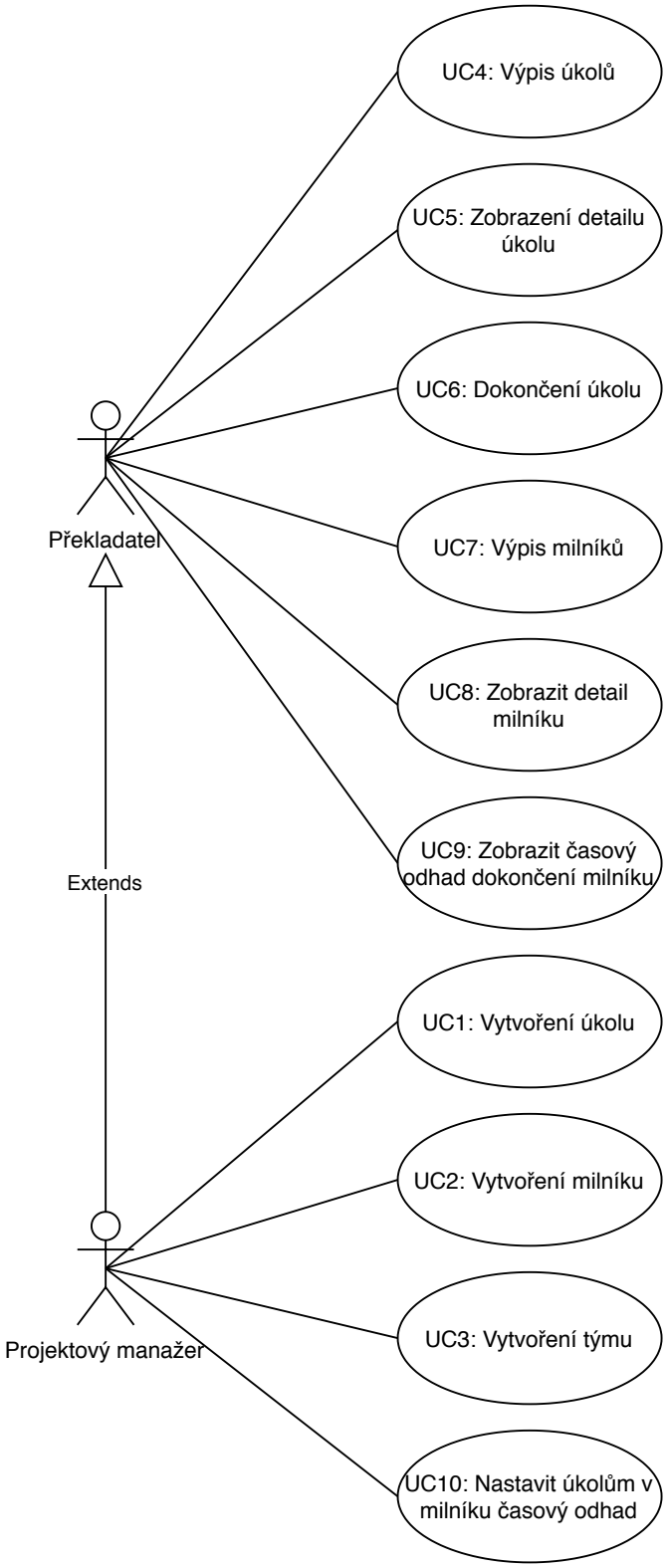
1. Případ užití začíná při kliknutí na tlačítko „Vytvořit úkol” v navigačním panelu nebo ve výpisu úkolů.

2. Systém zobrazí obrazovku s formulářem pro vytvoření úkolu.

3. Uživatel zadá povinné a volitelné parametry.

4. Svůj výběr uživatel potvrdí stiskem tlačítka „Vytvořit úkol” přímo pod formulářem.

5. Aplikace vytvoří úkol podle parametrů zadaných uživatelem a případ užití končí.



Obrázek 3.1: UML diagram případů užití

UC2: Vytvoření milníku

Umožňuje uživateli s oprávněním projektový manažer vytvořit milník. Uživatel musí povinně zadat název a popis milníku. Volitelně pak může vybrat úkoly, které do milníku spadají.

1. Případ užití začíná při kliknutí na tlačítko „Vytvořit milník" v navigačním panelu nebo ve výpisu milníků.
2. Systém zobrazí obrazovku s formulářem pro vytvoření milníku.
3. Uživatel zadá povinné parametry a volitelně vybere úkoly z výpisu úkolů, které do milníku spadají.
4. Svůj výběr uživatel potvrdí stiskem tlačítka „Vytvořit milník" přímo pod formulářem.
5. Aplikace vytvoří milník podle parametrů zadaných uživatelem a případ užití končí.

UC3: Vytvoření týmu

Umožňuje uživateli s oprávněním projektový manažer vytvořit tým. Uživatel musí zadat název a popis týmu. Volitelně může vybrat uživatele, kteří do týmu budou patřit.

1. Případ užití začíná při kliknutí na tlačítko „Vytvořit tým."
2. Systém zobrazí obrazovku s formulářem pro vytvoření týmu.
3. Uživatel zadá povinné parametry a volitelně vybere uživatele, kteří do týmu budou patřit.
4. Svůj výběr uživatel potvrdí stiskem tlačítka „Vytvořit tým" přímo pod formulářem.
5. Aplikace vytvoří tým podle parametrů zadaných uživatelem a případ užití končí.

UC4: Výpis úkolů

1. Případ užití začíná, když uživatel klikne na tlačítko „Úkoly."
2. Systém zobrazí obrazovku s listem všech úkolů.
3. Případ užití končí.

UC5: Zobrazení detailu úkolu

1. Případ užití začíná, když uživatel klikne na tlačítko „Zobrazit" u úkolu v listu všech úkolů z případu užití UC4: Výpis úkolů.
2. Systém na stejné obrazovce rozšíří pole, kde se nachází vybraný úkol, směrem dolů. Na rozšířeném poli jsou vidět všechny atributy úkolu.
3. Případ užití končí.

UC6: Dokončení úkolu

1. Případ užití začíná, když uživatel klikne na tlačítko „Dokončit" u úkolu v listu všech úkolů z případu užití UC4: Výpis úkolů.
2. Systém označí úkol v databázi jako dokončený a na obrazovce se u úkolu zobrazí ukazatel, že je dokončený.
3. Případ užití končí.

UC7: Výpis milníků

1. Případ užití začíná, když uživatel klikne na tlačítko „Milníky."
2. Systém zobrazí obrazovku s listem všech milníků.
3. Případ užití končí.

UC8: Zobrazení detailu milníku

1. Případ užití začíná, když uživatel klikne na tlačítko „Zobrazit" u milníku v listu všech milníků z případu užití UC7: Výpis milníků.
2. Systém na stejné obrazovce rozšíří pole, kde se nachází vybraný milník, směrem dolů. Na rozšířeném poli jsou vidět všechny atributy úkolu.
3. Případ užití končí.

UC9: Zobrazit časový odhad dokončení milníku

Aby byl případ užití zpřístupněn, je zapotřebí, aby všechny úkoly v milníku měly nastavený časový odhad.

1. Případ užití začíná, když uživatel klikne na tlačítko „Časový odhad" v detailu milníku z případu užití UC8: Zobrazení detailu milníku.
2. Systém zobrazí obrazovku s Ganttovým diagramem. Na Ganttovu diagramu budou vyobrazeny jednotlivé úkoly v závislosti na jejich prerekvizitách a časových odhadech. Více o vykreslení Ganttova diagramu v sekci 3.1.5.

3. Případ užití končí.

UC10: Nastavit úkolům v milníku časový odhad

Uživatel s rolí Projektový správce může pomocí počítače nastavit časový odhad všem úkolům, které ho nemají. Odhad realizovaný počítačem je nepřesný, ale napomůže získat rychlejší odhad dokončení milníku.

1. Případ užití začíná, když uživatel klikne na tlačítko „Nastavit úkolům v milníku časový odhad" v detailu milníku z případu užití UC8: Zobrazení detailu milníku.

2. Systém následně pro všechny úkoly obsažené v milníku, které nemají časový odhad, nastaví hodnotu vypočítanou v závislosti na obsaženém počtu klíčů a nastaveném koeficientu času na jeden klíč.

3. Případ užití končí.

3.3 Wireframy

3.3.1 Použitý nástroj

Pro vytvoření wireframů pro návrh rozšíření podpory projektové správy Weblate bylo použito Adobe XD. Tento nástroj byl vybrán kvůli autorově preferenci. Adobe XD je zdarma dostupný na [54].

3.3.2 Design a rozvrh obrazovek

Jelikož je Weblate již existující projekt, obrazovky byly navrženy podle designu Weblate, aby lépe reflektovaly finální produkt. Finální design se ale nejspíše bude lišit. Při designu byly rozšířeny navigační prvky, které jsou v aplikaci Weblate již nyní – to by mělo způsobit snadnější použitelnost a seznámení s novými funkcionalitami pro již zaběhlé uživatele Weblate. Aplikace bude uniformní co se týče rozložení i designu. Rozložení, které je použito, je jednoduché na používání a poskytuje dostatečnou schopnost orientovat se mezi přidanými funkcionalitami.

3.3.3 Obrazovky

V této subsekci budou uvedeny a popsány všechny důležité obrazovky. Měly by dostatečně reprezentovat finální vzhled, jak pro stakeholdery, tak pro běžné uživatele Weblate. V přiložených souborech je dostupný soubor Adobe XD, který obsahuje wireframy v interaktivní podobě. Zároveň jsou wireframy podporující interakci dostupné online, na [55].

Rozložení rozšířené výchozí stránky projektu Celkový design byl převzat z Weblate. Na rozložení a designu wireframu na obrázku 3.2 lze vidět, jak byly rozšířeny základní ovládací prvky. V černé navigační liště, která se týká nastavení celé Weblate aplikace, byla přidána sekce „Týmy," kde se po kliknutí zobrazí drop-down menu, přes které se dá dostat na obrazku s vytvořením týmu. Do budoucna mohou být přidány další možnosti jako například „Seznam týmů."

Do ovládacího panelu se zelenými tlačítky, který se týká vždy specifického projektu, byly přidány dvě sekce. Jedna se soustředí na milníky a je možné se přes ní dostat na obrazovku s výpisem milníků a na obrazovku s vytvořením milníku nového.

Druhá sekce je orientovaná na úkoly. Z drop-down menu je možné se dostat na obrazovku obsahující výpis úkolů a na obrazovku určenou pro vytvoření nového úkolu.

Ostatní komponenty wireframu zůstaly identické s originálními.

Přehled milníků Obrazovka 3.3 reprezentuje hlavně případ užití UC6 3.2. Je zobrazen přehled milníků s jejich časovým odhadem a indikátorem hotovosti. Detail každého milníku lze zobrazit při kliknutí na příslušné tlačítko „Zobrazit." Navíc bylo přidáno více viditelné tlačítko pro přidání nového milníku.

Detail milníku Lze vidět na obrázku 3.4. Detail milníku se zobrazí v rozšířeném poli příslušného milníku. K již zobrazeným atributům se objeví popis milníku a úkoly obsažené milníku.

Pokud mají obsažené úkoly nastavené časové odhady, je zobrazen časový odhad a po kliknutí na tlačítko „Časový rozvrh" je možné zobrazit Ganttův diagram. Co vše bere Ganttův diagram v potaz, je popsáno v sekci 3.1.5.

Pokud nemají všechny obsažené úkoly nastaven časový odhad, je možné pomocí tlačítka „Nastavit úkolům v milníku časový odhad" těmto úkolům nastavit časový odhad, který se počítá v závislosti na komponentech, kterých se úkol týká. O tomto výpočtu více v sekci 3.1.5.

Obrazovka s Ganttovým diagramem Po kliknutí na tlačítko „Časový odhad"se zobrazí Ganttův diagram vykreslený podle popisu v sekci 3.1.5. Lze vidět na obrázku 3.5.

Vytvoření nového milníku Obrazovka 3.6 slouží k vytvoření nového milníku. Formulář pro vytvoření milníku si zachovává stejné rozložení a design jako jiné formuláře ve Weblate. Ve formuláři je možné zadat všechny atributy milníku.

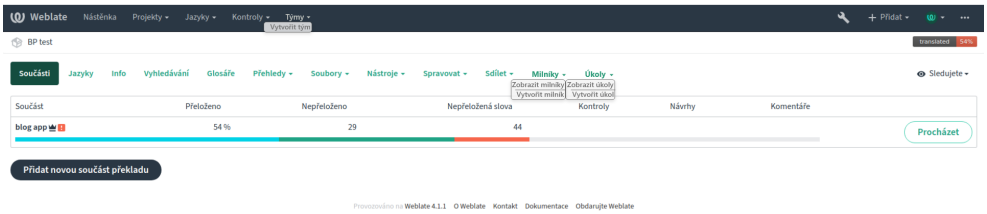
Přehled úkolů Obrazovka 3.7 reprezentuje hlavně případ užití UC4 3.2. Na obrazovce je zobrazen přehled úkolů s jejich prioritou, konečným termínem a časovým odhadem. Detail každého úkolu lze zobrazit při kliknutí na příslušné

tlačítko „Zobrazit.” Navíc bylo přidáno více viditelné tlačítko pro přidání nového úkolu.

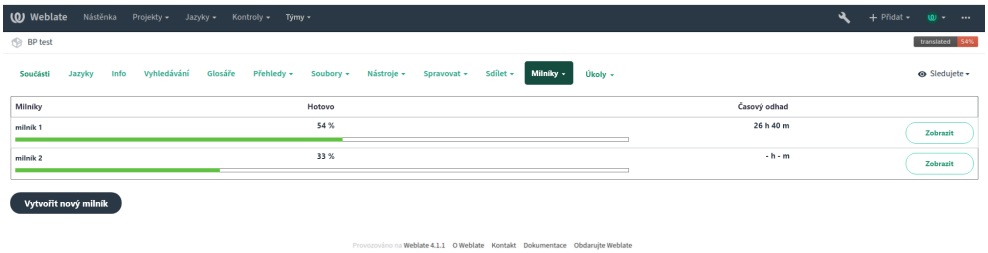
Detail úkolu Pokud se na obrazovce s přehledem úkolů klikne na tlačítko „Zobrazit,” rozšíří se pole s úkolem směrem dolů a zobrazí všechny atributy úkolu. Lze vidět na obrázku 3.8.

Vytvoření nového úkolu Obrazovka 3.9 slouží k vytvoření nového úkolu. Formulář pro vytvoření úkolu si zachovává stejné rozložení a design jako jiné formuláře ve Weblate. Ve formuláři je možné zadat všechny atributy úkolu.

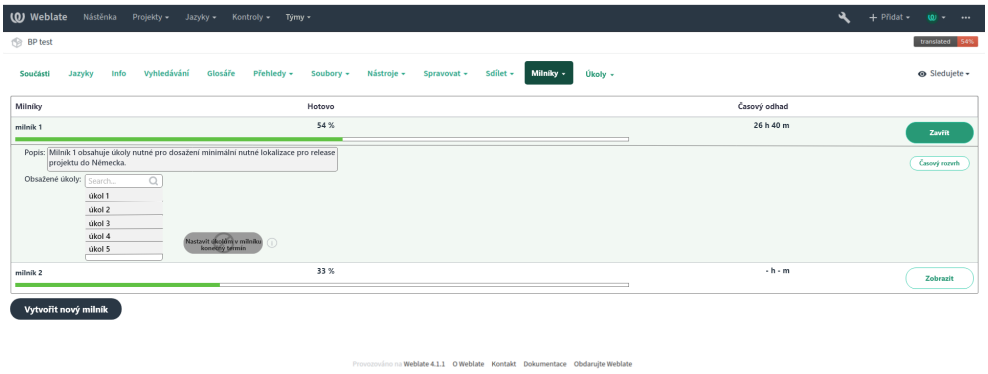
Vytvoření nového týmu Obrazovka 3.10 slouží k vytvoření nového týmu. Formulář pro vytvoření týmu si zachovává stejné rozložení a design jako jiné formuláře ve Weblate. Ve formuláři je možné zadat všechny atributy týmu.



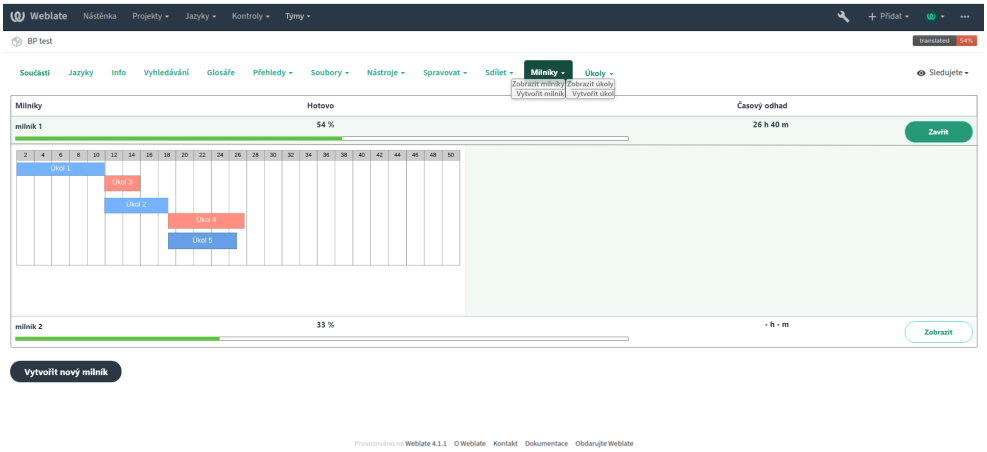
Obrázek 3.2: Wireframe – výchozí stránka projektu



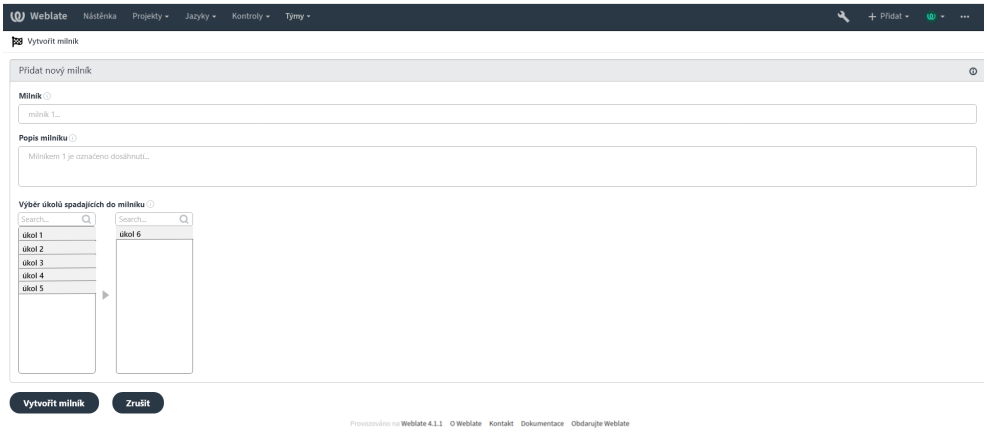
Obrázek 3.3: Wireframe – přehled milníků



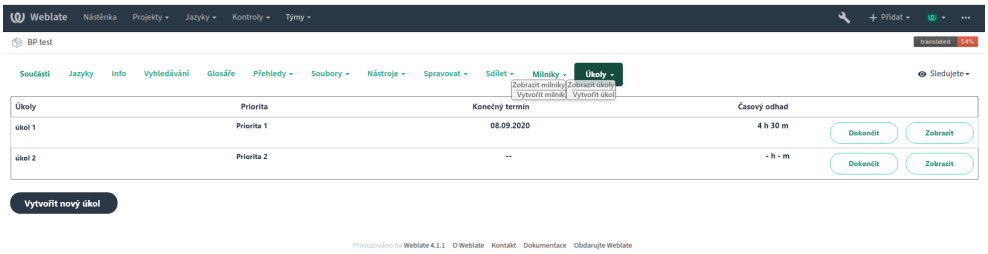
Obrázek 3.4: Wireframe – detail milníku



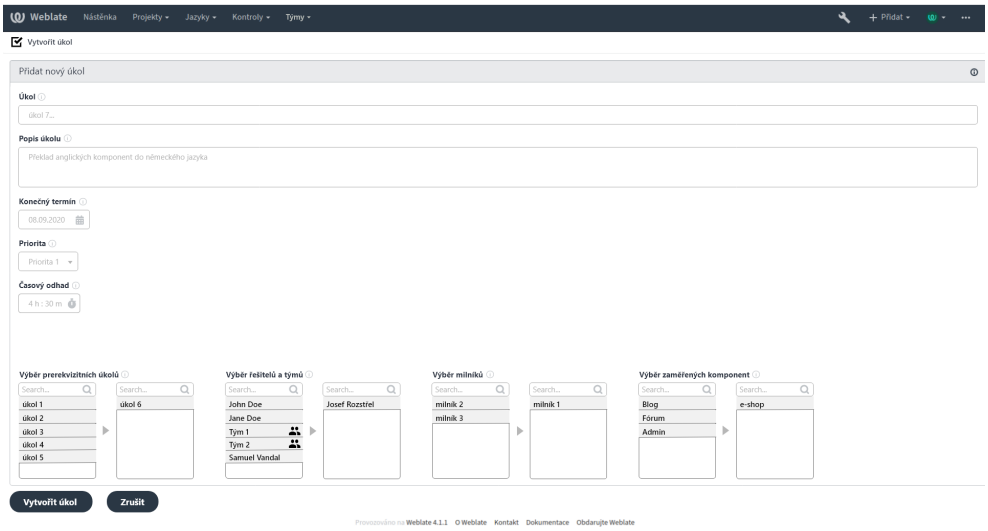
Obrázek 3.5: Wireframe – časový odhad



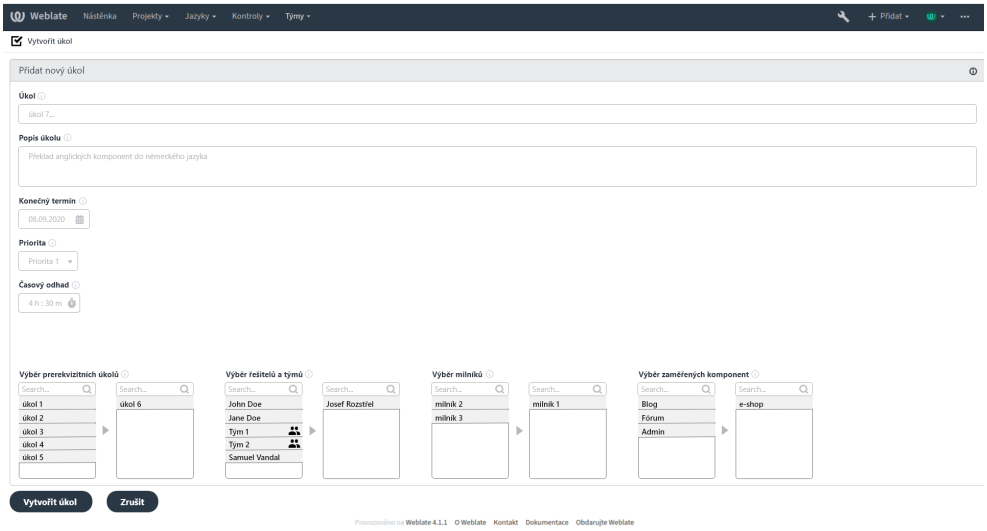
Obrázek 3.6: Wireframe – vytvoření nového milníku



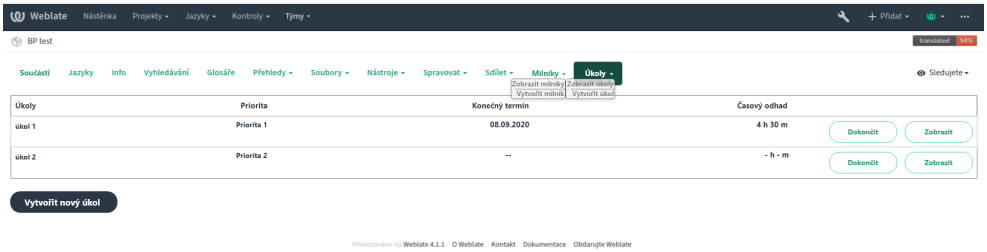
Obrázek 3.7: Wireframe – přehled úkolů



Obrázek 3.8: Wireframe – detail úkolu



Obrázek 3.9: Wireframe – vytvoření nového úkolu



Obrázek 3.10: Wireframe – vytvoření nového týmu

Kapitola 4

Strategická analýza

4.1 SWOT analýza

Z analýzy vyjdou najevo výhody, nevýhody, rizika a příležitosti implementace funkcionality projektového řízení do aplikace Weblate. Některé z otázek pokládaných pro jednotlivé aspekty jsou následující.

Silné stránky Čím nové funkcionality přidají hodnotu Weblate? Co je jiné oproti ostatním řešením, co tyto funkcionality mají?

Slabé stránky Co důležitého funkcionality postrádají? Jakou nevýhodu můžou nové funkcionality přinést Weblate? Je Weblate dostatečně novodobý a do budoucna udržitelný nástroj?

Příležitosti Jaké trendy mohou pozitivně působit? Uspokojí funkcionality pro projektovou správu nějaké potřeby na trhu? Mají konkurenti Weblate slabiny, které je možné nahradit? Je uživateli Weblate tato funkcionalita žádaná?

Rizika Může přijít změna na trhu, která by tuto funkcionality pro projektovou správu učinila zbytečnou? Co se může stát, aby Weblate přestal být užívaný?

4.1.1 Silné stránky

- Jelikož je Weblate používaný open source projekt, má zároveň i velké množství přispívajících programátorů.
- Nové funkcionality významně rozšíří už nyní objemný seznam funkcionalit.
- Aplikace bude nově podporovat projektovou správu.

- Projektová správa lokalizace bude zakomponována v lokalizačním systému. Nebude muset být separátně.

4.1.2 Slabé stránky

- Weblate je open source projekt, funkcionality budou nejspíše muset být implementovány dobrovolně.
- Nové funkcionality možná nebudou dostatečné rozsáhlé pro projektové manažery.

4.1.3 Příležitosti

- Lokalizace je nyní velmi důležitá pro firmy. Poptávka po řešeních nejspíše bude stoupat.
- Weblate bude ze všech analyzovaných řešení zdarma jediné, které bude projektovou správu podporuje.
- Alternativy Weblate mají nižší hodnotu.
- Open source řešení získávají na popularitě.
- Weblate má již nyní rozsáhlou uživatelskou základnu. Bude jednoduché propagovat nové funkcionality.
- Rozšířenou projektovou správu podporuje z analyzovaných řešení zatím pouze Lokalise.

4.1.4 Rizika

- Funkcionality nebudou používané dostatečně uživateli, aby se vyplatily implementovat.
- Vyjde nové řešení, či se vylepší existující řešení, které nabídne funkcionality pro projektovou správu dříve, než se funkcionality do Weblate implementují.
- Dojde k technologické změně a lokalizace pomocí překladových katalogů už nebude používaná.

4.2 Marketingová strategie

4.2.1 Cílová skupina

Cílovou skupinu tvoří 3 skupiny. Přehled skupin je v tabulce 4.1. Největší z nich je skupina uživatelů, která již Weblate užívá. Tuto skupinu by přidání funkcionality mělo pouze utvrdit v užívání Weblate namísto jiného řešení.

Druhá skupina jsou uživatelé jiných lokalizačních řešení. Cílem je tuto skupinu přesvědčit k přesunutí na Weblate.

Třetí skupinou jsou potenciální zákaznící, kteří žádný lokalizační systém nepoužívají. Cílem je přesvědčit je, že Weblate je dostatečně hodnotné řešení a vyplatí se vynaložit zdroje na jeho užívání.

Skupina 1	Uživatelé Weblate
Skupina 2	Uživatelé jiných lokalizačních řešení
Skupina 3	Potenciální uživatelé lokalizčních řešení

Tabulka 4.1: Cílové skupiny Weblate.

4.2.2 Marketingový plán

Cílová skupina 1 Plánem je tuto skupinu pouze utvrdit v tom, že Weblate je lokalizační řešení s nejvyšší hodnotou a odradit je od případné změny řešení.

Cílové skupiny 2 a 3 V plánu je vytvořit marketingovou kampaň na sociálních sítích, aby se využilo faktu, že Weblate je již existující řešení se stávající uživatelskou základnou. Jelikož projekt nemá za hlavní cíl vydělávat, placené marketingové strategie nejsou plánovány.

Kapitola 5

Ekonomická a právní analýza

5.1 Odhad práce

Tabulka 5.2 uvádí hrubý odhad, kolik času by stál vývoj nových funkcionalit pro rozšířenou podporu projektového řízení. Vytvoření Ganttova diagramu bylo zváženo, ale jelikož není předem znám počet vývojářů, kteří se budou na vývoji podílet, Ganttův diagram neposkytuje v tomto případě žádnou přidanou hodnotu. Všechny odhady jsou v člověkodnech (množství práce, kterou může udělat člověk za jeden den, tedy 8 hodin).

Předpokládá se, že vývojář je již zkušený a zdatný v tom, co je zapotřebí k implementaci funkcionalit. Jelikož je to první odhad, musíme předpokládat vysoký faktor nejistoty. Obvykle to znamená, že skutečný čas strávený realizací projektu bude v rozmezí 0,25—4 krát náročnější oproti původnímu odhadu. S více investovaným časem do projektu se nepřímou úměrou i zužuje kužel nejistoty, což umožňuje přesnější odhady [56].

Předpoklady byly založeny na návrhu základních případů užití společně se zkušenostmi autora v softwarovém inženýrství a s vývojem webových aplikací v Django frameworku. Pro vytvoření odhadů byly zvoleny t-shirt odhady, vysvětleny v sekci 1.2. Tabulka 5.1 definuje velikosti triček.

Velikost:	**Odhad**
S	0–1 ČD
M	1–2 ČD
L	2–3 ČD
XL	3–5 ČD
XXL	5–8 ČD

Tabulka 5.1: Tabulka s definicí velikostí triček.

	Odhad
Softwarový návrh:	**3–7 ČD**
Definice zbylých případů užití	S
Třídový diagram	M
Návrh architektury nových funkcionalit	M
Grafický návrh	M
Implementace:	**11–19 ČD**
Implementace nových modelů	L
Migrace databáze	S
Implementace Weblate Django aplikace na projektové řízení	XL
Implementace šablon	L
Stylování šablon	M
Vytvoření nových testů	XL
Testování a dokumentace:	**10–16 ČD**
Všeobecné testování	XL
Testování běžnými uživateli	XXL
Doplnění dokumentace	L
Nasazení změn:	**1–2 ČD**
Merge do hlavního repozitáře	M
Celkem:	**25–44 ČD**

Tabulka 5.2: Tabulka odhadu práce.

5.2 Odhad ceny implementace

Projekt je určen pro implementaci jedním vývojářem nebo malým týmem (2—3 osoby). Náklady nebudou odhadnuty ze dvou důvodů. Za prvé je těžké odhadnout náklady, a to i v širokém rozsahu, jelikož hodinová sazba se velmi liší v závislosti na senioritě vývojáře.

Za druhé se jedná o open source projekt, tudíž se také předpokládá, že funkcionalita bude implementována někým dobrovolně v rámci dobrého úmyslu. Z těchto důvodů nebyl proveden detailní odhad nákladů. Uvedené odhady práce by měly být dostačující pro studii proveditelnosti.

Pokud by byl zapotřebí odhad nákladů implementace pro nacenění práce, v tabulce 5.3 je velmi hrubý odhad ceny práce za předpokladu dostatečné seniority vývojářů a analytiků. Pro vytvoření odhadu byl použit odhad práce ze sekce 5.1 a data o platech v České republice získaná z [57].

Pracovní pozice:	**Odhad**
Medior IT analytik	8 000–18 000 Kč
Medior Python developer	61 000–102 000 Kč
Celkem:	**69 000–120 000 Kč**

Tabulka 5.3: Hrubý odhad nákladů.

5.3 Provozní náklady

Jelikož se přidané funkcionality implementují do Weblate aplikace pouze jako další funkcionality, provozní náklady zůstanou stejné jako dříve.

5.4 Přínos projektu

Jak je popsáno v sekci 2.1. Tento projekt není realizován se záměrem vydělat, ale se záměrem přispět odvětví lokalizace a celkově napomoci procesu globalizace, což je i hlavním přínosem projektu. Projekt nemá za cíl peněžní přínos.

5.5 Právní analýza

Weblate využívá veřejně dostupné vývojové technologie, které jsou zdarma k užití. Weblate je vyvinut hlavně v Django frameworku, který podléhá 3-bodové BSD licenci. Finální produkt je tedy možné využívat i komerčně. Přidané funkcionality se nemusí zabývat zákonem o ochraně dat, jelikož žádná osobní data nebudou zpracovávat. Všechna právní ustanovení Weblate zůstanou stejná jako doposud.

Závěr

Cílem této bakalářské práce bylo vytvoření studie proveditelnosti vylepšení řešení jazykové lokalizace webových aplikací v jazyce Python.

Při tvorbě práce proběhla důkladná analýza šesti již existujících řešení jazykové lokalizace a bylo vyhodnoceno, že nejlepším dosavadním řešením je Weblate, který je nejlepší téměř ve všech hodnocených aspektech, které byly definovány na začátku práce. Na základě této analýzy bylo rozhodnuto, že stávající nejlepší řešení je dostačující, ale chybí mu funkcionalita podporující projektové řízení, která se ve dvou jiných řešeních vyskytuje a přidá Weblate významně hodnotu.

Na teoretické úrovni byly navrženy hlavní případy užití, které by bylo zapotřebí implementovat, a wireframy, které dobře imitují finální uživatelskou interakci a poskytnou přehled o tom, jak by funkcionality mohly fungovat. Na základě teoretického návrhu bude v budoucnu možné zefektivnit proces implementace a designu.

Byla provedena strategická analýza obsahující SWOT analýzu, která zajišťuje přehled silných a slabých stránek projektu, a návrh marketingové strategie.

Provedena byla i ekonomická analýza zahrnující odhad práce, odhad ceny implementace, diskuzi provozních nákladů a možný finální přínos projektu. Poskytuje tak přehled o nákladech na realizaci projektu. Právní aspekt projektu byl také prozkoumán.

Celkově tato práce obsahuje dostatek informací pro dokázání, že vylepšení řešení jazykové lokalizace webových aplikací v Pythonu je proveditelné. Zároveň může tato práce sloužit jako dokumentace při implementaci navrženého řešení.

Bibliografie

1. ESSELINK, Bert. *A practical guide to localization*. Benjamins, 2000. ISBN 9789027219558.
2. KENTON, Will. Feasibility Study. In: *Investopedia* [online] [cit. 2020-07-09]. Dostupné z: `https://www.investopedia.com/terms/f/feasibility-study.asp`.
3. SIEBER, Patrik. Studie proveditelnosti (feasibility study). *Metodická příručka. Ministerstvo pro místní rozvoj. Praha*. 2004. Dostupné také z: `https://www.dotaceeu.cz/getmedia/c4772855-8ffc-4036-97fc-2d7caa1ad86e/1136372156-zpracov-n-studie-proveditelnosti.pdf`.
4. Steps to Prepare Feasibility Study Report for Software Development Project. In: *Project practical* [online] [cit. 2020-07-22]. Dostupné z: `https://www.projectpractical.com/steps-to-prepare-feasibility-study-report-for-software-development-project/`.
5. Types of Feasibility Study in Software Project Development. In: *Geeks for geeks* [online] [cit. 2020-07-22]. Dostupné z: `https://www.geeksforgeeks.org/types-of-feasibility-study-in-software-project-development/`.
6. BIN, Yu. A Novel Method of Real Estate Development Project's Feasibility Research Based on SWOT Method and Analytic Hierarchy Process. *International Journal of Business and Social Science*. 2014, roč. 5, č. 5. Dostupné také z: `http://ijbssnet.com/journals/Vol_5_No_5_April_2014/29.pdf`.
7. HILL, Terry; WESTBROOK, Roy. SWOT analysis: it's time for a product recall. *Long range planning*. 1997, roč. 30, č. 1, s. 46–52. Dostupné z DOI: `10.1016/S0024-6301(96)00095-7`.
8. DESS, Gregory. *Strategic management: Text and cases*. McGraw-Hill Education, 2013. ISBN 9780077862527.

9. Project Estimation Through T-Shirt size. In: *Medium* [online] [cit. 2020-07-25]. Dostupné z: https://medium.com/radius-engineering/project-estimation-through-t-shirt-size-ea496c631428.

10. HELDMAN, Kim. *PMP: project management professional exam study guide*. John Wiley & Sons, 2018. ISBN 9781119420910.

11. INSTITUTE, Project Management. *A guide to the project management body of knowledge (PMBOK Guide)*. Project Management Inst, 2000. ISBN 9781880410233.

12. BOOCH, Grady. *The unified modeling language user guide*. Pearson Education India, 2005. ISBN 9780321267979.

13. RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. The unified modeling language. *Reference manual*. 1999. ISBN 9780201309980.

14. JACOBSON, Ivar. *Object-oriented software engineering: a use case driven approach*. Pearson Education India, 1993. ISBN 9788131704080.

15. ROSENBERG, Doug; SCOTT, Kendall. *Use case driven object modeling with UML*. Springer, 1999. Dostupné z DOI: 10.1007/978-1-4302-0369-8_3.

16. Origin of Use Case. In: *Visual paradigm* [online] [cit. 2020-07-12]. Dostupné z: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/.

17. BROWN, Dan M. *Communicating design: developing web site documentation for design and planning*. New Riders, 2010. ISBN 9780131385412.

18. GARRETT, Jesse James. *The elements of user experience: user-centered design for the web and beyond*. Pearson Education, 2010. ISBN 9780321624635.

19. What is wireframing? In: *Experience UX* [online] [cit. 2020-07-20]. Dostupné z: https://www.experienceux.co.uk/faqs/what-is-wireframing/.

20. KUHLMAN, Dave. *A python book: Beginning python, advanced python, and python exercises*. Dave Kuhlman Lutz, 2009. Dostupné také z: https://www.davekuhlman.org/python_book_01.pdf.

21. OLIPHANT, Travis E. Python for scientific computing. *Computing in Science & Engineering*. 2007, roč. 9, č. 3, s. 10–20. Dostupné také z: http://www.orender.net/scholar/thesis/python-scientific-computing.pdf.

22. DocForge - Web application framework. In: *Web archive* [online] [cit. 2020-07-02]. Dostupné z: https://web.archive.org/web/20150723163302/http://docforge.com/wiki/Web_application_framework.

23. Top Open Source Static Site Generators. In: *StaticGen* [online] [cit. 2020-07-02]. Dostupné z: https://www.staticgen.com/.

24. Django FAQ. In: *Django documentation* [online] [cit. 2020-07-02]. Dostupné z: https://docs.djangoproject.com/en/dev/faq/general/%5C#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names.

25. Design philosophies. In: *Django documentation* [online] [cit. 2020-07-03]. Dostupné z: https://docs.djangoproject.com/en/2.0/misc/design-philosophies/.

26. What Powers Instagram: Hundreds of Instances, Dozens of Technologies. In: *Instagram engineering* [online] [cit. 2020-07-03]. Dostupné z: https://instagram-engineering.com/what-powers-instagram-hundreds-of-instances-dozens-of-technologies-adf2e22da2ad.

27. Django Web Framework (Python). In: *Mozzila infrastructure web documentation* [online] [cit. 2020-07-03]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django.

28. Open Source at The Washington Times. In: *Washington times* [online] [cit. 2020-07-03]. Dostupné z: http://opensource.washingtontimes.com/.

29. What is Translation Management? In: *SDL Trados* [online] [cit. 2020-07-09]. Dostupné z: https://www.sdltrados.com/solutions/translation-management/.

30. DREPPER, Ulrich; MEYERING, Jim; PINARD, Francois; HAIBLE, Bruno. GNU gettext tools, version 0.20. 2. 1995. Dostupné také z: https://www.gnu.org/software/gettext/manual/gettext.pdf.

31. GROSS, Steffen et al. Internationalization and localization of software. *Eastern Michigan University, Department of Computer Science, Ypsilanti, Michigan, Thesis June*. 2006, roč. 19. Dostupné také z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.175.2883&rep=rep1&type=pdf.

32. How to translate Python Applications with GNU gettext. In: *Phrase* [online] [cit. 2020-07-09]. Dostupné z: https://phrase.com/blog/posts/translate-python-gnu-gettext/.

33. The Format of PO Files. In: *GNU website* [online] [cit. 2020-07-09]. Dostupné z: https://www.gnu.org/savannah-checkouts/gnu/gettext/manual/html_node/PO-Files.html.

34. Gettext - Multilingual internationalization services. In: *Python documentation* [online] [cit. 2020-07-09]. Dostupné z: https://docs.python.org/3/library/gettext.html.

35. Translation. In: *Django documentation* [online] [cit. 2020-07-09]. Dostupné z: https://docs.djangoproject.com/en/3.0/topics/i18n/translation/.

36. Django admin makemessages. In: *Django documentation* [online] [cit. 2020-07-09]. Dostupné z: `https://docs.djangoproject.com/en/3.0/ref/django-admin/%5C#django-admin-makemessages`.

37. I18n. In: *Django documentation* [online] [cit. 2020-07-09]. Dostupné z: `https://docs.djangoproject.com/en/3.0/topics/i18n/`.

38. Rosetta repozitář. In: *GitHub* [online] [cit. 2020-07-09]. Dostupné z: `https://github.com/mbi/django-rosetta`.

39. Translation manager. In: *COEX* [online] [cit. 2020-07-09]. Dostupné z: `https://www.coex.cz/blog/translation-manager`.

40. DTM repozitář. In: *GitHub* [online] [cit. 2020-07-09]. Dostupné z: `https://github.com/COEXCZ/django-translation-manager`.

41. Lokalise web. In: *Lokalise* [online] [cit. 2020-07-09]. Dostupné z: `https://lokalise.com/`.

42. Lokalise GitLab integration. In: *Lokalise documentation* [online] [cit. 2020-07-09]. Dostupné z: `https://docs.lokalise.com/en/articles/1789855-gitlab`.

43. Lokalise Pricing. In: *Lokalise* [online] [cit. 2020-07-09]. Dostupné z: `https://lokalise.com/pricing`.

44. Zanata platform repozitář. In: *GitHub* [online] [cit. 2020-07-09]. Dostupné z: `https://github.com/zanata/zanata-platform`.

45. Zanata web. In: *Zanata* [online] [cit. 2020-07-09]. Dostupné z: `http://zanata.org/`.

46. Zanata Server repozitář. In: *GitHub* [online] [cit. 2020-07-09]. Dostupné z: `https://github.com/zanata/zanata-docker-files/tree/master/zanata-server`.

47. Weblate web. In: *Weblate* [online] [cit. 2020-07-09]. Dostupné z: `https://weblate.org/cs/`.

48. Continuous localization. In: *Weblate documentation* [online] [cit. 2020-07-09]. Dostupné z: `https://docs.weblate.org/en/latest/admin/continuous.html`.

49. Addons. In: *Weblate documentation* [online] [cit. 2020-07-09]. Dostupné z: `https://docs.weblate.org/en/latest/admin/addons.html`.

50. ZCommand hooks. In: *Zanata documentation* [online] [cit. 2020-07-09]. Dostupné z: `http://docs.zanata.org/en/release/client/command-hook/`.

51. Django packages ranking. In: *Django packages* [online] [cit. 2020-07-09]. Dostupné z: `https://djangopackages.org/grids/g/i18n/`.

52. LARSON, Erik W; GRAY, Clifford F. Project management: The managerial process. 2011. ISBN 9780071289290.

53. CLELAND, David I.; KING, William R. Project management handbook. In: 1983.

54. Adobe XD download. In: *Adobe* [online] [cit. 2020-07-17]. Dostupné z: `https://www.adobe.com/products/xd.html`.

55. Project management wireframes. In: *Adobe XD* [online] [cit. 2020-07-20]. Dostupné z: `https://xd.adobe.com/view/5e48035f-29e6-4879-a1d4-a55f3610dbb0-0b1b/?fullscreen`.

56. The Cone of Uncertainty. In: *Construx* [online] [cit. 2020-07-17]. Dostupné z: `https://www.construx.com/books/the-cone-of-uncertainty/`.

57. Průzkum platů - platy podle pozic. In: *Platy* [online] [cit. 2020-07-25]. Dostupné z: `https://www.platy.cz/platy`.

Příloha **A**

Seznam použitých zkratek

TMS Translation management system

UML Unified modeling language

I18N Internationalization

L10N Localization

PO Portable object

POT Portable object template

URL Uniform resource locator

DTM Django translation manager

SaaS System as a service

CLI Command-line interface

VCS Version control system

QA Quality assurance

UC Use case

Příloha **B**

Obsah přiložené SD karty

- `README.md` ..... stručný popis obsahu SD karty
- `thesis.pdf` ..... práce ve formátu PDF
- `src`
  - `repository.zip` ..... Git repozitář s testovací aplikací
  - `thesis.zip` ..... zdrojová forma práce ve formátu LaTeX
  - `Wireframes_weblate_PM.xd` ..... wireframy ve formátu Adobe XD