FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Rozšíření Timer2Ticket
Student:	Bc. Jan Starůstka
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Manažerská informatika
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem práce je vhodně rozšířit již existující synchronizační nástroj (middleware) Timer2Ticket, jehož současnou verzi vyvíjeli v rámci svých závěrečných prací Ing. Vít Štefan, Bc. Jakub Čermák a Bc. Martin Paul. Práce bude zaměřena na další projektové nástroje.

1. Analyzujte stávající stav aplikace Timer2Ticket a projektové nástroje, o které by bylo vhodné aplikaci rozšířit. Dále analyzujte zatížení společnosti Jagu s.r.o. s ohledem na synchronizaci mezi projektovými nástroji, které současně využívají.
2. Proveďte vhodný sběr a vhodnou analýzu požadavků potenciálních i aktuálních uživatelů.
3. Na základě analýzy proveďte důkladný návrh potřebných změn a rozšíření tohoto synchronizačního nástroje.
4. Dle návrhu implementujte vámi vhodně zvolená rozšíření.
5. Navrhněte a realizujte vhodné sady testů s důrazem na bezproblémové budoucí použití a provoz služby / rozšíření.
6. Vyhodnoťte přínos Implementovaných změn na vytížení zdrojů společnosti Jagu s.r.o. s ohledem na synchronizaci dat mezi projektovými nástroji.
7. Zhodnoťte dosažené výsledky a navrhněte možná další budoucí rozšíření.

Elektronicky schválil/a Ing. David Buchtela, Ph.D. dne 7. února 2024 v Praze.

Diplomová práce

ROZŠÍŘENÍ TIMER2TICKET

Bc. Jan Starůstka

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Hunka
4. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především vedoucímu práce, Ing. Jiřímu Hunkovi za odborné vedení a pomoc při řešení problémů. Dále bych chtěl poděkovat všem, kteří mi pomohli při sběru požadavků a uživatelském testování: Ing. Oldřich Malec, Sára Sovičková, Bc. Jakub Čermák a MUDr. Adam Šercl. Děkuji také Ing. Martinu Paulovi za know-how, které mi předal z jeho předchozí práce na Timer2Ticket. Všichni jste mi velmi pomohli k lepšímu výsledku.
Mnohokrát děkuji své rodině a přátelům za podporu nejen při tvorbě diplomové práce, ale především za jejich dlouhodobou podporu po celou dobu mého studia.

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 4. května 2024

Abstrakt

Ve své práci jsem se věnoval rozšíření synchronizačního nástroje Timer2Ticket. Především jsem se zaměřil na přidání projektového nástroje Jira. Nejdříve jsem analyzoval požadavky a provedl výběr nástroje k implementaci, následně jsem navrhl způsob zakomponování požadavků do již existujícího kódu a databáze. Většinu z navržených změn jsem následně implementoval, podrobil uživatelským testům a zpracoval vyhodnocení dopadu na využití zdrojů společnosti Jagu s. r. o. Na základě zpětné vazby od uživatelů jsem navrhl úpravy uživatelského rozhraní a nastínil možný postup při dalším rozšiřování nástroje Timer2Ticket.

Klíčová slova Timer2Ticket, Jira, Redmine, Toggl Track, synchronizace časových záznamů, synchronizace ticketovacích nástrojů, webhooks, middleware, JavaScript, TypeScript, Vue.js, MongoDB, REST API

Abstract

In this thesis I discussed possible extension of synchronization middleware Timer2Ticket. I mainly focused on addition of project tool Jira. I began with requirements analyses and selection of a tool to add into Timer2Ticket aplication. I designed possible implementation of such requirements and implemented most of them. Impact of implemented changes was studied on Jagu s. r. o. company. Usability tests of the web user interface were performed on the application afterwards. Based on user feedback, I proposed user interface improvements. Finally I summarized my recommendations for the future developement of the Timer2Ticket application.

Keywords Timer2Ticket, Jira, Redmine, Toggl Track, Synchronization of Time Entries, Synchronization of Ticket Applications, webhooks, middleware, JavaScript, TypeScript, Vue.js, MongoDB, REST API

Seznam zkratek

API	Application Programming Interface
BPMN	Business Process Model and Notation
DB	Databáze
JS	JavaScript
JSON	JavaScript Object Notation
JQL	Jira Query Language
STEO	Service Time Entry Object
SW	Software
T2T	Timer2Ticket
TESO	Time Entry Synced Object
TS	TypeScript
UC	Use Case (případ užití)
URL	Uniform Resource Locator

Kapitola 1
Úvod

Aplikace Timer2Ticket slouží k synchronizaci časových záznamů mezi nástroji pro měření času a projektovými nástroji. Pod nástrojem pro měření času si můžeme představit stopky, které zapínáme a vypínáme podle práce na jednotlivých úkolech, které dostáváme zadané. Takovým nástrojem je například Toggl Track. Úkoly dostáváme zadané v projektovém nástroji, například Jira, nebo Redmine. Tyto nástroje umožňují kromě zadávání úkolů sledovat a řídit projekty. Proto je vhodné vědět, kolik který úkol zabere jednotlivým členům týmu času.

Vývojáři bývají často na projektech úkolováni přes Issues/Tickets (úkoly). Jedná se o jednoduchou a srozumitelnou formu komunikace, která podporuje předání jasného zadání vývojáři. Vývojář pak na úkolu pracuje a hotový ho vrátí zadavateli ke kontrole. Často je možné, dokonce i povinné připojit k úkolu informaci o čase potřebném k jeho realizací. To je dobré pro monitoring a plánování projektu.

Členové týmu mají několik možností, jak čas strávený na jednotlivých úkolech měřit. Od kvalifikovaných odhadů až po nástroje pro tento úkol specificky určené. Projektový manažer samozřejmě chce mít co nejpřesnější data. Proto bude vývojáře tlačit do použití vhodných nástrojů. Navíc bude po vývojářích vyžadovat, aby se skutečný čas realizace úkolu dostal do jeho projektového nástroje, ve kterém jsou vedeny jednotlivé úkoly. Manuální přenášení dat je vždy potenciálním zdrojem chyb a nepřesností, navíc je jím tráveno nezanedbatelné množství času, který lze investovat lépe.

Řešením těchto problémů je nástroj Timer2Ticket, který synchronizuje časové záznamy mezi aplikací určenou k měření času a projektovým nástrojem, ve kterém je řízen projekt a zadávány úkoly. Timer2Ticket vytvoří vývojářům v aplikaci pro měření času kopie úkolů, které byly zadané v projektovém nástroji. Vývojář pak jednoduše v aplikaci zvolí, na kterém úkolu zrovna pracuje a naměří tak čas, který ke splnění úkolu potřeboval.

Timer2Ticket následně automaticky synchronizuje tyto časové záznamy s časovými záznamy vedenými v projektovém nástroji. Projektový manažer tak získá data o časové náročnosti jednotlivých úkolů a o vytížení jednotlivých členů týmu bez toho, aby museli jednotliví vývojáři tato data manuálně přenášet mezi aplikacemi.

Nástroj Timer2Ticket pomáhá i v případech, kdy vývojář pracuje na více projektech, případně pro více zákazníků. Je totiž schopen z nástroje pro měření času zjistit, ke kterému projektu a zákazníkovi patří který časový záznam. To následně usnadní vývojářům práci při fakturaci.

S vývojem nástroje Timer2Ticket začal ve své diplomové práci Ing. Vít Štefan [1]. Ten implementoval synchronizaci časových záznamů mezi měřičem času Toggl Track [2] a projektovým nástrojem Redmine [3]. Navázal na něj ve své bakalářské práci Bc. Jakub Čermák [4], který navrhl nové webové rozhraní a způsob monetizace aplikace. Implementaci nového webového klienta a dalších úprav navržených Jakubem Čermákem provedl ve své diplomové práci Ing. Martin Paul [5].

V této práci se budu zabývat dalším vhodným rozšířením nástroje Timer2Ticket, především o další projektový nástroj a propojení tohoto nového nástroje s aktuálně implementovaným nástrojem pro měření času Toggl Track a projektovým nástrojem Redmine. Rozšířením o další projektový nástroj bude Timer2Ticket schopen uspokojit potřeby více uživatelů a tím pádem přiláká i více platících zákazníků.

Kapitola 2
Analýza

V této kapitole se věnuji analýze současného stavu aplikace Timer2Ticket. Analyzuji stav aplikace pro navázání v práci na aplikaci po Vítu Štefanovi, Jakubu Čermákovi a Martinu Paulovi. Dále zkoumám a vybírám další projektové nástroje vhodné pro implementaci. Následně se věnuji sběru a analýze požadavků, především ohledně přidání dalšího projektového nástroje Jira.

2.1 Analýza současného stavu

Práci na téma synchronizačního middleware Timer2Ticket začal ve své diplomové práci Ing. Vít Štefan [1], navázal na něj svou bakalářskou prací Bc. Jakub Čermák [4] a do současného stavu aplikaci ve své diplomové práci dovedl Bc. Martin Paul [5].

2.1.1 Účel aplikace Timer2Ticket

Nástroj Timer2Ticket slouží k synchronizaci časových záznamů mezi nástroji pro měření času („Timer“) a projektovým nástrojem („Ticket“). Nástroje pro měření času slouží uživatelům pro přesný záznam času stráveného na jednotlivých úkolech. Ticketovací nástroj má zase za úkol přiřazovat úkoly jednotlivým členům týmu a celkově podporovat projektové řízení.

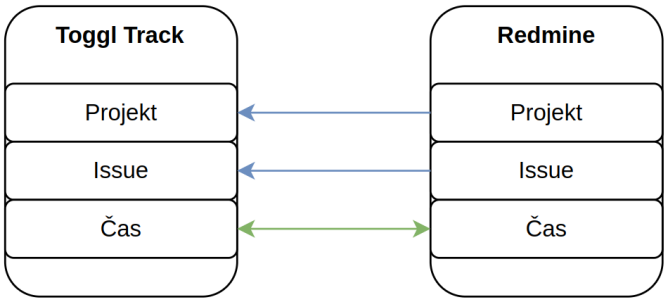
Přenášení údajů mezi těmito aplikacemi je časově náročné a náchylné na chyby. Timer2Ticket toto řeší tak, že v nástrojích pro měření času vytváří projekty a úkoly. K těm pak uživatel při měření přiřadí konkrétní časový záznam. Záznam je následně synchronizován zpět do projektového nástroje. Uživatel tak nemusí řešit časové výkazy, stačí když bude správně vybírat úkoly při měření času.

Pro koho je aplikace určená

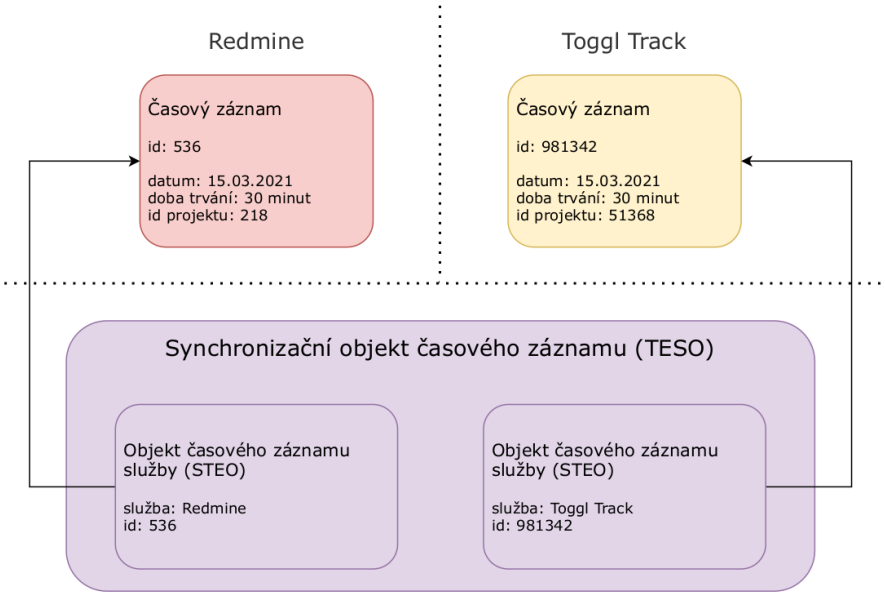
Aplikace Timer2Ticket je učena především pro softwarové vývojáře. Ti často využívají nějaký nástroj pro měření času (například Toggl Track [2]), ale úkoly dostávají zadané v jiném, ticketovacím (projektovém) systému (například Redmine [3]). Timer2Ticket jim umožňuje efektivněji využívat nástroj pro měření času díky synchronizaci úkolů a ušetří jim čas manuálního vykazování času díky synchronizaci časových záznamů.

Hlavní funkcionalita aplikace

Aplikace Timer2Ticket slouží především k synchronizaci časových záznamů mezi nástrojem pro měření času („Timer“) a projektovým nástrojem („Ticket“). V tuto chvíli podporuje synchro-



Obrázek 2.1 Synchronizace objektů mezi Redmine a Toggl Track



Obrázek 2.2 Objekty STEO a TESO, převzato z [1]

nizaci mezi měřičem času Toggl Track [2] a projektovým nástrojem Redmine [3]. Když uživatel zprovozní synchronizaci, budou se mu synchronizovat úkoly a projekty jedním směrem, z Redmine do Toggl Track, a oběma směry bude probíhat synchronizace časových záznamů. Schématicky je toto zaznačeno na obrázku 2.1. Šipky modrou barvou značí jednosměrnou synchronizaci z Redmine do Toggl Track a zelenou barvou je vyznačena obousměrná synchronizace času. Přidání nebo odebrání časového záznamu z jednoho systému tak symetricky upraví záznam i ve druhém systému.

Fungování synchronizace časových záznamů

Nástroj Timer2Ticket si v databázi udržuje pro každé spojení dvou nástrojů mapovací objekty TESO (Time Entry Synced Object) a STEO (Service Time Entry Object), viz obrázek 2.2. TESO obsahuje informace o nějakém časovém záznamu a kolekci objektů STEO. Každé STEO odkazuje na časový záznam v synchronizované službě. Detailněji popisuje fungování nástroje Timer2Ticket v kapitole 2.6 ve své práci Vít Štefan [1].

Spojením se zde rozumí synchronizace dvou instancí aplikace. Například Toggl Track běží pouze na jedné instanci, ale projektový nástroj Redmine si může každý provozovat sám na jiné adrese. Ke konfiguraci spojení je od uživatele třeba získat API klíč a workspace z Toggl Track

a API klíč, API point a výchozí aktivita z Redmine. Výchozí aktivita bude využita ve chvíli, kdy uživatel u změřeného času v Toggl Track nevybral žádný konkrétní úkol. Čas je pak přidán pouze k projektu a ke zvolené aktivitě.

2.1.2 Předplatné nástroje Timer2Ticket

Nástroj Timer2Ticket byl ve své původní verzi od Víta Štefana [1] bezplatný a umožňoval pouze jedno spojení mezi nástrojem Toggl Track a Redmine. Monetizační model vytvořil ve své bakalářské práci Jakub Čermák [4]. Ten navrhl tři členství: Hobby, Junior a Senior.

Hobby členství je zdarma a umožňuje mít aktivní pouze jedno spojení mezi nástroji a synchronizaci pouze jednou týdně. Junior členství za $5 měsíčně nabízí uživateli 2 aktivní spojení a synchronizace bude probíhat jednou denně. Uživatel se Senior členstvím za $8 má možnost využívat až 5 aktivních spojení a synchronizace bude probíhat každou hodinu. Nad rámec členství lze dokoupit okamžité synchronizace, nebo další.

Členství a různé frekvence synchronizací do nástroje Timer2Ticket implementoval ve své práci Martin Paul [5].

2.1.3 Architektura nástroje Timer2Ticket

Aplikace Timer2Ticket se skládá z několika nezávislých částí znázorněných na obrázku 2.3. Pro uživatele je zde webový interface, ten komunikuje s vrstvou Api, která se stará o zpracování uživatelových požadavků a o autentizaci. Core vrstva provádí samotnou synchronizaci mezi synchronizovanými službami. API i Core komunikují se stejnou databází MongoDB. Vrstva Api v databázi konfiguruje spojení vrstva a Core podle dat v databázi provádí synchronizaci.

Timer2Ticket komunikuje s několika dalšími externími službami. Kromě synchronizovaných nástrojů jsou to ještě platební brána Stripe [6] a fakturační nástroj Fakturoid [7].

Frontend Timer2Ticket

Původní frontend navrhl a realizoval ve své Diplomové práci Vít Štefan [1]. Byl velice jednoduchý a umožňoval nastavit pouze jedno spojení mezi Redmine a Jira. Neumožňoval přihlášení přes Auth0[8] a nebyl přizpůsobený na monetizační model dle [4]. Původní verze aplikace vůbec neřešila členství, bylo proto možné synchronizovat nástroje kdykoliv na vyžádání uživatele. Náhled starého frontend si můžete prohlédnout na obrázku 2.4. Starý frontend byl implementován v jazyce TypeScript [9] s využitím frameworku Angular[10].

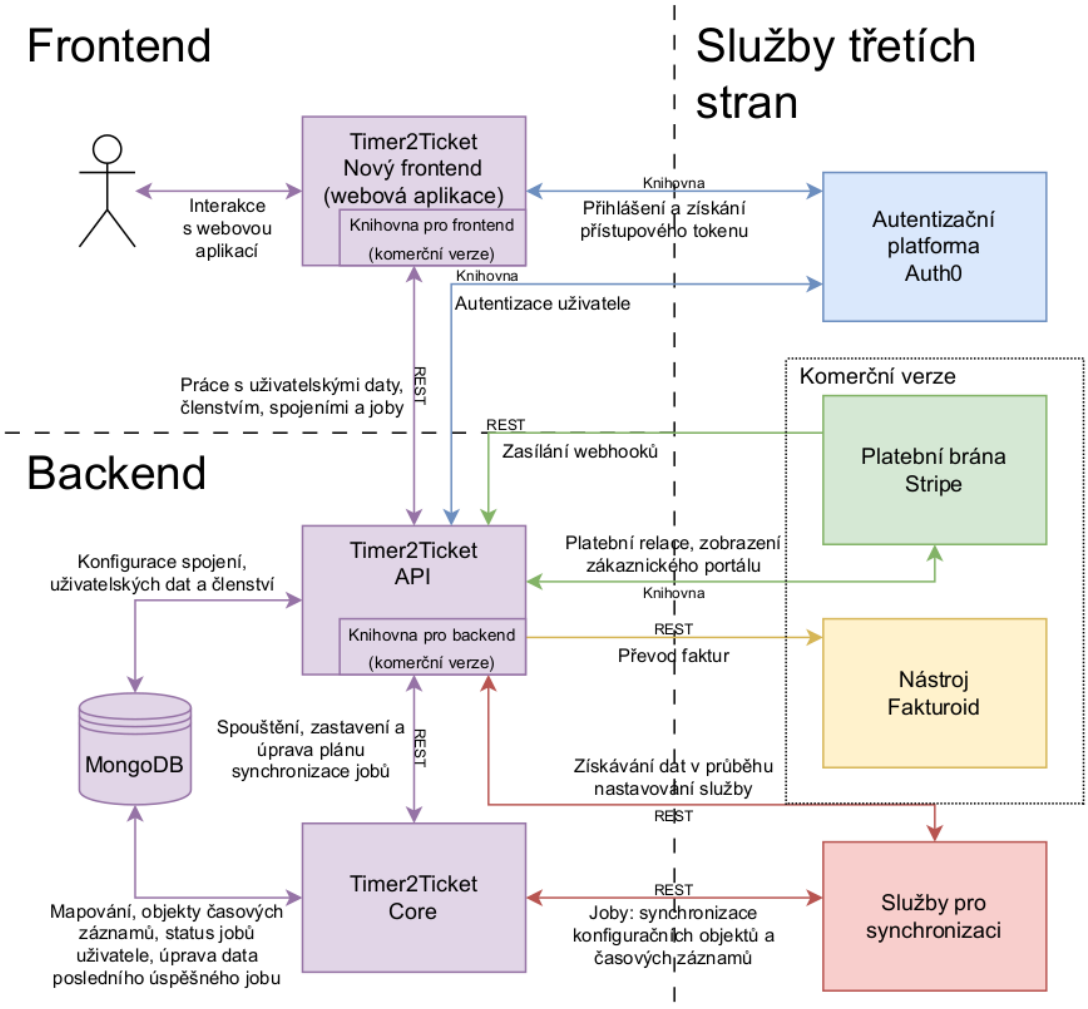
Nový frontend navrhl ve své práci Jakub Čermák [4] a s původním backendem spojil ve své práci Martin Paul [5]. Je implementován v jazyce JavaScript [11] s využitím knihovny Vue.js [12]. Nový frontend podporuje přihlašování pře Auth0 [8], umožňuje uživateli mít více spojení a pracuje s různými předplatnými služby Timer2Ticket a je připravený na rozšíření o další nástroje pro synchronizaci (kromě Toggl Track a Redmine).

Timer2Ticket Api

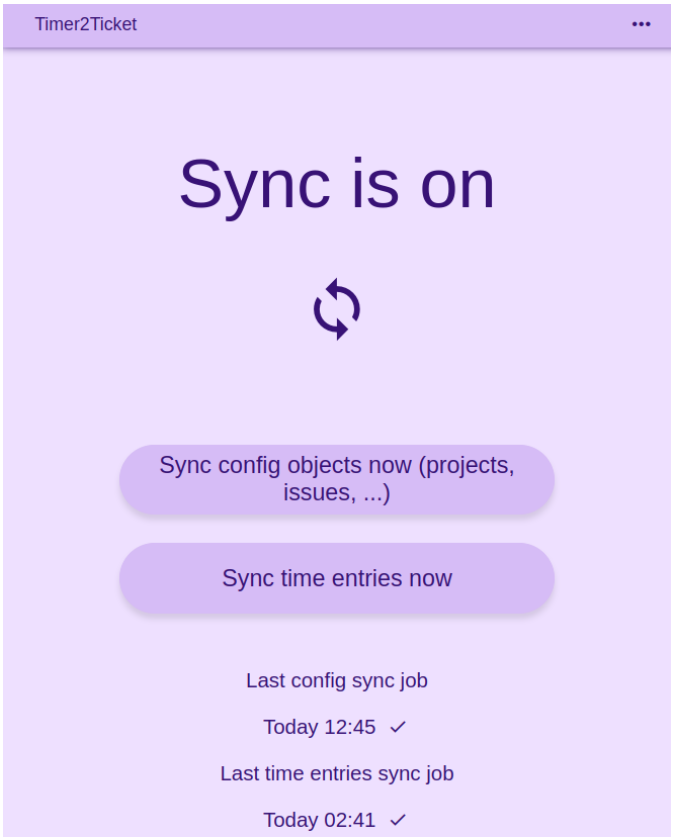
Vrstva Api přijímá požadavky od klienta a nastavuje konfiguraci synchronizací vrstvy Core. Navíc řeší například autorizaci požadavků. S API je možné komunikovat přes REST API [13]. Implementováno je v jazyce TypeScript [9] a běží na platformě Node.js [14] s využitím frameworku Express.js [15] a knihoven SuperAgent [16], Class Validator [17] a Node Cron [18].

Timer2Ticket Core

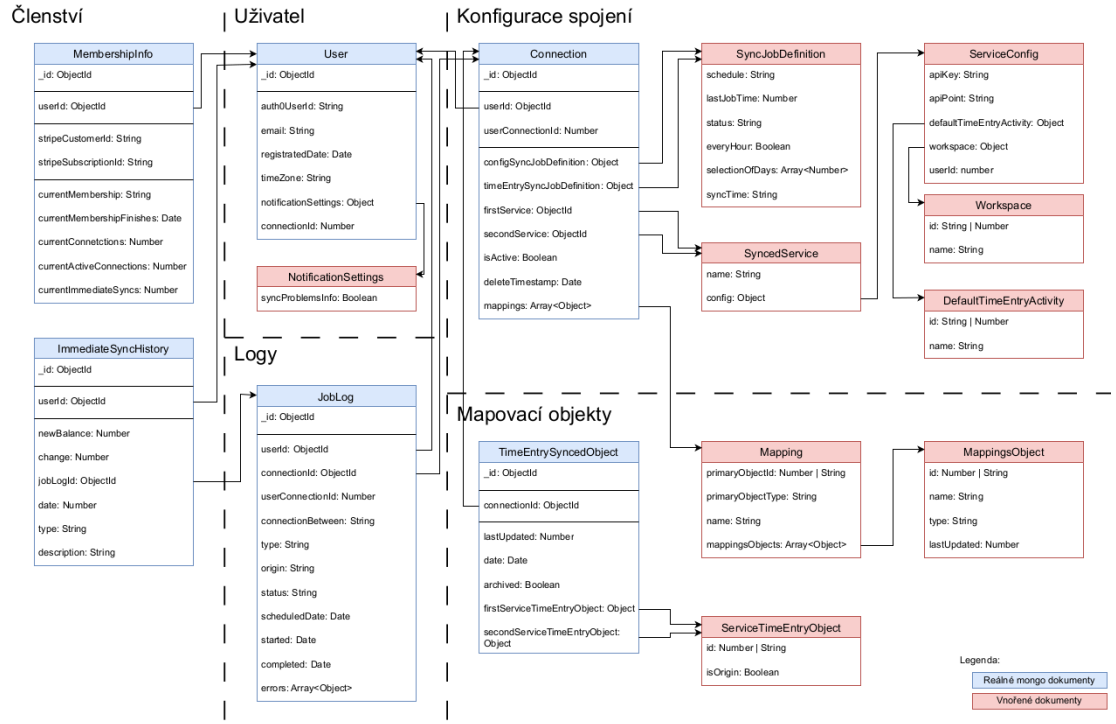
Core (jádro) nástroje Timer2Ticket se stará o samotnou synchronizaci jednotlivých objektů (projekty a úkoly) a časových záznamů. V pravidelných intervalech (nebo na vyžádání) spouští joby,



Obrázek 2.3 Architektura nástroje Timer2Ticket, převzato z [5]



Obrázek 2.4 Starý frontend nástroje Timer2Ticket



Obrázek 2.5 Schéma Databáze, bez notace, převzato z [5]

které provádějí synchronizaci. Co a jak jednotlivé joby provádí vysvětluje Vít Štefan v kapitole 2.6 [1].

Job synchronizace konfigurace – Spravuje a synchronizuje projekty a úkoly mezi službami.

Job synchronizace časových záznamů – Synchronizuje časové záznamy mezi službami.

Úklidový job – Promazává staré a nepotřebné objekty, například staré logy ostatních jobů.

Vrstva Core má vystavené REST API, přes které s ním komunikuje vrstva Timer2Ticket Api.

Databáze

Timer2Ticket využívá dokumentovou databázi MongoDB [19]. V původním návrhu Víta Štefana (kapitola 2.8 [1]) měla databáze 3 kolekce: User, TimeEntrySyncedObject a Job Log. V rámci implementace monetizačního modelu v práci Martina Paula (kapitola 3.6 [5]) došlo ale k drobnému přepracování schématu databáze a k přidání dalších kolekcí.

V nekomerční verzi má databáze nyní čtyři kolekce. Přibyla nová kolekce *User*, která drží data o jednotlivých uživatelích. Kolekce *Connection* reprezentuje spojení vytvořené uživatelem. Kolekce *TimeEntrySyncedObject* reprezentuje páry časových záznamů a kolekce *JobLog* uchovává informace o proběhnutých jobech.

V komerční verzi má databáze Timer2Ticket navíc dvě kolekce. *MembershipInfo* uchovává data o členství uživatele a *ImmediateSyncHistory* zase drží informaci o historii okamžitých synchronizací. Obě tyto kolekce jsou úzce navázány na monetizační model. Aktuální databázový model z práce Martina Paula [5] je zobrazen na obrázku 2.5.

Autentizace a autorizace

V původní verzi Víta Štefana [1] byla autorizace a autentizace implementována pomocí JWT token [20]. V práci Jakuba Čermáka [4] vznikl požadavek na přihlašování přes platformu Auth0 [8], zejména kvůli umožnění registrace přes sociální sítě a další služby třetích stran. Autorizaci a autentizaci přes Auth0 implementoval ve své práci Martin Paul [5].

Platební brána Stripe

Platební brána umožňuje přijímat platby od zákazníků. Ti si díky tomu mohou pořídit lepší členství i okamžité synchronizace a spojení nad rámec členství.

Jakub Čermák ve své práci [4] vybral jako vhodnou platební bránu Stripe [6]. Implementaci platební brány do komerční verze provedl ve své práci Martin Paul [5]. Většina proběhlých plateb jsou pravidelné v režimu předplatného. Proto zde byla vhodná implementace pomocí webhooks, kdy platební brána Stripe notifikuje aplikaci Timer2Ticket o příchozích platbách.

Nástroj Fakturoid

Po obdržení informace o zaplacení od platební brány je zákazníkovi vystavena faktura. K tomu je využit nástroj Fakturoid [7], kterému jsou zaslány informace o zákazníkovi a platbě přes API. Z těchto dat je následně vytvořena faktura, která zůstane uložená v nástroji Fakturoid pro provozovatele Timer2Ticket Jagu s. r. o. [21]

2.1.4 Možnosti rozšíření Timer2Ticket

V předchozích pracích na téma Timer2Ticket [1, 4, 5] zaznívaly požadavky na rozšíření o další projektové nástroje i aplikace pro měření času. Aktuální požadavky na integraci nového nástroje od Martina Paula (požadavky 23-25) mluví o integraci Jira podobným způsobem, jako je nyní do aplikace integrován Redmine. Navíc zmiňuje i novou „Enterprise“ verzi, která by zajistila synchronizaci celých projektů v různých aplikacích.

Možným směrem vývoje také může být přidání webhooks. To by umožnilo rychlejší reakci na změny v synchronizovaných aplikacích. Ne všechny nástroje ale webhooks podporují (Redmine). To může být problém při správném nastavování členství.

2.2 Analýza vhodných nástrojů k implementaci do Timer2Ticket

Timer2Ticket je aplikace synchronizující časové záznamy mezi více aplikacemi. Tyto aplikace jsou dvojího druhu: Aplikace pro měření času – „Timer“ (Toggl Track [2]) a projektové nástroje – „Ticket“ (Redmine [3]). Tato práce se zabývá integrací projektového nástroje, proto záměrně opomenu alternativy nástroje Toggl Track a budu se věnovat alternativám Redmine.

Už v předchozích pracích na Timer2Ticket [1, 4, 5] byl analyzován nástroj Jira [22], na který se zaměřím kvůli jeho velké oblibě [23] a na žádost zadavatele. Další alternativy byly vybrány na základě dat o jejich využívání dostupných z [24, 23, 25].

Toggl Track

Jedná se o nástroj pro měření času, který nabízí velmi intuitivní měření na jednotlivých projektech a úkolech. V nabídce je několik členství s tím, že v základním členství je k dispozici samotné měření a pohled na reporty z něj. Za příplatek je uživateli umožněno přidat k projektů svou hodinovou mzdu a zjednodušit tak následnou fakturaci.

Základní členství *Basic* je zdarma, lepší *Starter* členství stojí 10$ měsíčně a nejdražší *Premium* 20$ za měsíc. Pro zprovoznění synchronizace na projektový nástroj Jira je nutné mít prémiové členství.

Analýzu Toggl Track i jeho implementaci do Timer2Ticket provedl již Vít Štefan v rámci své diplomové práce [1]. Časové záznamy jsou synchronizovány z Toggl Track do uživatelem zvoleného projektového nástroje (v této chvíli pouze Redmine) pomocí štítků (Tags). Ty jsou v Toggl Track vytvořeny podle existujících úkolů v projektovém nástroji.

Jira

Jira [22] je komerční projektový nástroje vyvíjeny společností Atlassian [26]. Nabízí bezplatný plán, i několik placených plánů, které nabízí více funkcí. Uživateli pomůže s přípravou projektu a výběrem vhodné metodiky řízení projektu. Té následně přizpůsobí workflow, ve kterém budou spravovány úkoly. U jednotlivých úkolů je možné definovat nespočet vlastních atributů, včetně toho pro strávený čas.

Jira nabízí API, přes které lze úkoly modifikovat. Navíc je možné použití webhooks [27], což může být pro Timer2Ticket vhodné pro rychlé získávání dat o změnách v úkolech a časových záznamech. Už v základním plánu zdarma poskytuje Jira všechny potřebné věci pro případné připojení Timer2Ticket, dražší plány pak umožňují uživatelům lépe monitorovat a řídit projekty.

Podle [24, 23, 28] využívají Jira firmy všech velikostí, převážně v IT sektoru a většina z nich sídlí v anglicky mluvících zemích. Jira využívá 206 314 společností. Jedná se tak podle této statistiky o nejpoužívanější z analyzovaných nástrojů pro řízení projektu.

GitHub

GitHub [29] není projektovým nástrojem v pravém slova smyslu. Jedná se primárně o nástroj pro verzování kódu. Umožňuje ale v rámci jednotlivých repozitářů pracovat s issues (úkoly). Nativně k nim neumožňuje měřit čas, ale existuje mnoho nástrojů třetích stran, díky kterým to možné je, viz [30]. Jedná se ale především o připojení aplikací podobným Toggl Track, takže pokud bychom chtěli GitHub zakomponovat do Timer2Ticket, pravděpodobně by to znamenalo pozměnit koncept aplikace. V případě GitHubu by totiž nebyl čas veden v „Ticket“ nástroji, ale v „Timer“ nástroji. Synchronizace nástrojů by pak znamenala pouze stahování jednotlivých úkolů a projektů z GitHubu do nástroje na měření času, protože časové záznamy by nebylo kam do GitHubu synchronizovat.

S GitHub issues lze pracovat pomocí API a také je zde možnost využití webhooks. Chybí zde custom fields, které lze částečně suplovat pomocí labels. GitHub kromě plánu zdarma nabízí i placené verze, ty ale nepřidávají funkcionality relevantní pro aplikaci Timer2Ticket.

GitHub je dle [24, 23, 31] nejvíce rozšířen mezi firmami v IT sektoru. Data tvrdí, že GitHub využívá 203 464 firem, zejména do 200 zaměstnanců. Většina těchto firem funguje ve Spojených státech amerických.

GitLab

Podobně jako Github je i GitLab [32] nástrojem primárně pro verzování kódu a také v něm lze pracovat s issues u jednotlivých projektů (repozitářů). Navíc ale lze zaznamenávat čas přímo k jednotlivým issues bez nutnosti nějakého rozšíření, nebo aplikace třetí strany. GitLab poskytuje API, přes které je možné s issues pracovat a modifikovat tak strávený čas.

Issues lze modifikovat přes API a lze využívat webhooks. Nastavení vlastních atributů issues (custom fields) není možné. U GitLabu je běžná praxe, že si jej uživatel provozuje na vlastní infrastruktuře, proto pro tuto práci není relevantní jejich cenová nabídka plánů.

Podle dat [24, 23, 33] využívá GitLab 75 165 společností převážně v IT sektoru.

Asana

Asana[34] umožňuje komplexní správu projektu a dobré škálování na velikost týmu. Nabízí *Basic* členství zdarma pro týmy do 15 lidí a placené předplatné *Premium* a *Business*. Už v základním členství je možné spravovat úkoly. Ovšem až v premiovém členství je možné pracovat s custom fields a měření času je k dispozici dokonce až v nejdražší verzi Business. Přes API je možné spravovat úkoly (tasks) a lze využít webhooks.

Data z [24, 23, 35] evidují 52 724 společností využívajících Asana. Kromě IT sektoru je Asana populární především v odvětví marketingu. Využívají ji hlavně firmy s 10-200 zaměstnanci.

Smartsheet

Smartsheet [36] ve verzi zdarma umožňuje správu projektu pouze pro 3 lidi, dražší plány pak nabízí mnoho pomocníků pro nastavování workflow, monitorování a řízení projektů. U úkolů je možné měřit čas a zakládat custom fields již v plánu zdarma. Přes API Smartsheet je možné pracovat s úkoly a možné je i využití webhooks.

Smartsheet využívá 38 579 společností [24, 23, 37] převážně v IT. Na rozdíl od předchozích nástrojů je ale více populární i v jiných odvětvích, například medicíně. Smartsheet využívají společnosti všech velikostí. Oproti ostatním nástrojům má vyšší procentuální zastoupení mezi velkými společnostmi (200 a více lidí).

2.3 Výběr projektového nástroje k realizaci

Pro výběr vhodného projektového nástroje k implementaci do Timer2Ticket jsem stanovil hodnotící kritéria. Kritéria budou mít různá bodová ohodnocení podle důležitosti. Jednotlivé nástroje pak dostanou body podle splnění kritérií. Hodnocení na základě získaných bodů bude mít váhu 50%. Výběr nástroje zadavatelem Jagu s. r. o. bude mí taktéž váhu 50%. Součtem těchto dvou metrik bude jasné, který nástroj bude v rámci této práce implementován a také případné další pořadí nástrojů k implementaci do Timer2Ticket pro navazující práce.

2.3.1 Hodnotící kritéria nástrojů

Hodnotící kritéria byla volena na základě analýzy předchozích prací a podle požadavků na aplikaci od jednotlivých stakeholderů projektu Timer2Ticket ze sekce 2.5.

Měření času v rámci úkolu

Timer2Ticket stojí na tom, že v projektovém nástroji lze měřit čas strávený na úkolech. Proto je důležité, aby projektový nástroj umožňoval vykazování času k jednotlivým úkolům. Pokud nástroj dostane v tomto kritériu 0 bodů, bude hodnocen nulou v celkovém součtu.

2 body – V projektovém nástroji je možné měřit čas u úkolu ve verzi zdarma.

1 bod – Pro měření času u úkolu je třeba mít placenou verzi nástroje.

Vhodné API pro modifikaci úkolů

Bez API nebude možné rozumně přenášet data mezi systémy. Proto je z pohledu nástroje Timer2Ticket podstatné, aby projektový nástroj vystavoval API pro manipulaci s úkoly. Pokud nástroj dostane v tomto kritériu 0 bodů, bude hodnocen nulou v celkovém součtu.

1 bod – Projektový nástroj nabízí API pro správu úkolů.

	Jira	GitHub	GitLab	Asana	Smartsheet
Měření času v rámci úkolu	2	0	2	1	2
Vhodné API pro modifikaci úkolů	1	0	1	1	1
Podpora webhooks	1	1	1	1	1
Custom fields pro úkoly	2	0	2	1	2
Množství firem využívající nástroj	6	4	2	1	0
Suma bodů	12	0	8	5	6

Tabulka 2.1 Hodnocení projektových nástrojů pomocí zvolených kritérií

Podpora webhooks

Webhooks jsou vhodným nástrojem pro rychlé zjištění změn v synchronizovaném systému. Pokud služba vystavuje webhooks, umožní to uživatelům rychlejší synchronizace mezi jejich oblíbenými nástroji.

1 bod – Projektový nástroj podporuje použití webhooks.

Custom fields pro úkoly

Možnost definovat si vlastní pole u úkolů může být nezbytné pro správné spojení mezi jednotlivými systémy. Nelze navíc předpokládat stejné atributy úkolů napříč všemi potenciálními aplikacemi. Proto je pro nástroj Timer2Ticket i uživatele důležité, aby custom fields v nástroji šly vytvářet.

2 body – Projektový nástroj nabízí custom fields v rámci plánu zdarma.

1 bod – Projektový nástroj nabízí custom fields pouze v rámci placeného předplatného.

Množství firem využívající nástroj

Rozšíření Timer2Ticket o další projektový nástroj má mimo jiné za cíl zpřístupnit jeho používání pro více uživatelů. Čím více firem (a tím pádem i uživatelů) daný nástroj využívá, tím více potenciálních zákazníků může nástroj Timer2Ticket zajímat.

6 bodů – První v pořadí využívání

4 bodů – Druhý v pořadí využívání

2 body – Třetí v pořadí využívání

1 bod – Čtvrtý v pořadí využívání

0 bodů – Pátý v pořadí využívání

2.3.2 Hodnocení nástrojů

Hodnocení na základě kritérií

Nástroje byly hodnoceny na základě předchozích hodnotících kritérií. Jednotlivá hodnocení jsou zanesena do tabulky 2.1. Pokud nástroj nezískal body v některém z povinných kritérií, je celkově hodnocen nula body.

Nejvíce bodů získal nástroj Jira, druhý je nástroj GitLab, třetí Smartsheet a čtvrtá Asana. Poslední GitHub nesplňuje povinná kritéria a proto jej nyní není rozumné implementovat do nástroje Timer2Ticket.

Pořadí stanovené zadavatelem

Zadavatel seřadil projektové nástroje následovně:

1. Jira
2. Freelo
3. YouTrack
4. Asana
5. Smartsheet
6. GitLab
7. GitHub
8. Bitbucket
9. Trello

Protože jsem neuvažoval všechny ze zadavatelem preferovaných nástrojů, přidám zde ještě zběžnou analýzu Freelo a YouTrack. Trello a BitBucket také v mé analýze nefigurují, podle zadavatele mají ale nízkou prioritu a proto je z analýzy vynechám.

Freelo

Freelo [38] je český projektový nástroj. Má verzi zdarma i několik placených plánů lišících se maximální velikostí týmu a dostupnými funkcionalitami. Poskytuje správu projektů i úkolů a umožňuje i přímo v aplikaci (mobilní i webové) měřit čas už ve *Free* verzi. V Timer2Ticket by tak mohl fungovat jak v roli nástroje pro měření času, tak projektového nástroje.

Má k dispozici API pro práci s projekty, úkoly i časovými záznamy. API pro práci s časovými záznamy ovšem nabízí pouze 3 endpointy: start tracking, edit a end tracking. V současné verzi tedy chybí endpoint pro vytvoření časového záznamu i jeho mazání. Timer2Ticket by tak pro vytvoření časového záznamu musel nejdříve časomíru spustit a následně ji hned zastavit a správně zeditovat atributy podle potřeby. Jak realizovat případné mazání se mi nepodařilo dohledat.

Podle mnou definovaného bodového hodnocení by Freelo dostalo 5 bodů (viz tabulka).

YouTrack

YouTrack [39] je projektový nástroj, který podporuje správu projektů, úkolů a podporuje kolaboraci většího množství lidí a týmů. Všechny funkcionality nástroje jsou k dispozici už pro plán zdarma, který je ale omezen na 10 uživatelů. Za každého uživatele nad 10 je YouTrack zpoplatněný.

Úkolům v Youtrack lze přidávat libovolné custom fields a k úkolům je možné zaznamenávání času. Je zde velmi rozsáhlé API, včetně všeho, co by mohl Timer2Ticket k fungování potřebovat. YouTrack rovněž nabízí možnost konfigurace webhooks.

V hodnocení podle hodnotících kritérií získal YouTrack 6 bodů.

Přepočet pořadí od zadavatele na Body

V předchozích hodnotících kritériích 2.3.1 je možné získat maximálně 12 bodů. Tolik bodů získá první nástroj v pořadí podle zadavatele. Další nástroj v pořadí získá o bod méně atd.

	Freelo	YouTrack
Měření času v rámci úkolu	2	2
Vhodné API pro modifikaci úkolů	1	1
Podpora webhooks	1	1
Custom fields pro úkoly	1	2
Množství firem využívající nástroj	0	0
Suma bodů	5	6

Tabulka 2.2 Hodnocení dalších projektových nástrojů pomocí zvolených kritérií

	Bodové ohodnocení	Body od zadavatele	Celkem bodů	Pořadí
Jira	12	12	24	1
Freelo	5	11	16	2
YouTrack	6	10	16	2
Asana	5	9	14	4
Smartsheet	6	8	14	4
GitLab	8	7	15	3
GitHub	0	6	6	5

Tabulka 2.3 Hodnocení projektových nástrojů pomocí zvolených kritérií

2.3.3 Výběr nástroje

Výběr nástroje k implementaci byl vybrán spojením mnou definovaného bodového ohodnocení a pořadím od zadavatele. Celkové hodnocení je možné si prohlédnout v tabulce 2.3.

Vzhledem k výsledkům předchozích dvou hodnocení byl k implementaci v rámci této práce vybrán projektový nástroj **Jira**. Další v pořadí by případně byl nástroj Freelo, nebo YouTrack.

2.4 Aktuálně existující propojení Toggl Track a Jira

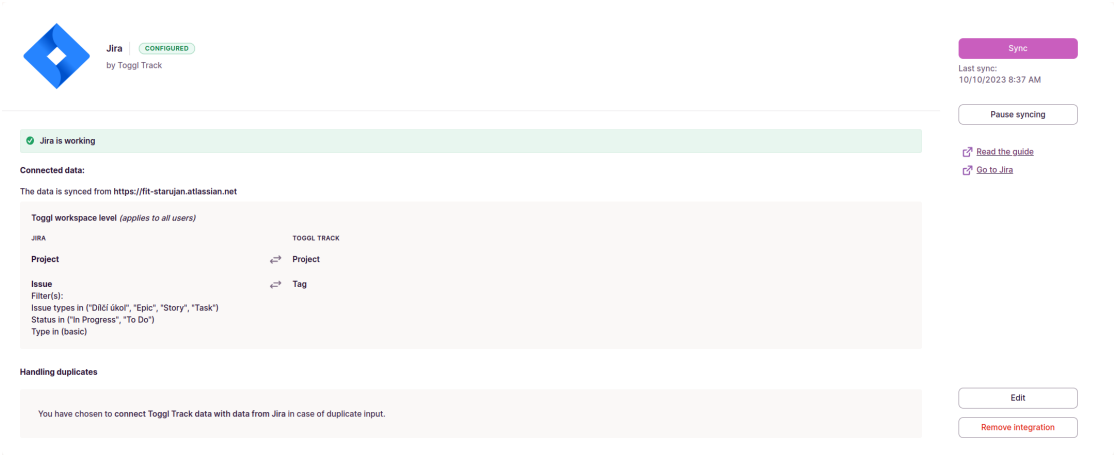
Pro lepší pochopení problému jsem ve své analýze věnoval čas již existujícím řešením spojení Jira a Toggl Track. Cílem je pochopit jak fungují a proč fungují právě takto. Následně se chci poučit z jejich silných i slabých stránek a navrhnout lepší synchronizaci v rámci nástroje Timer2Ticket.

2.4.1 Nativní propojení od Toggl

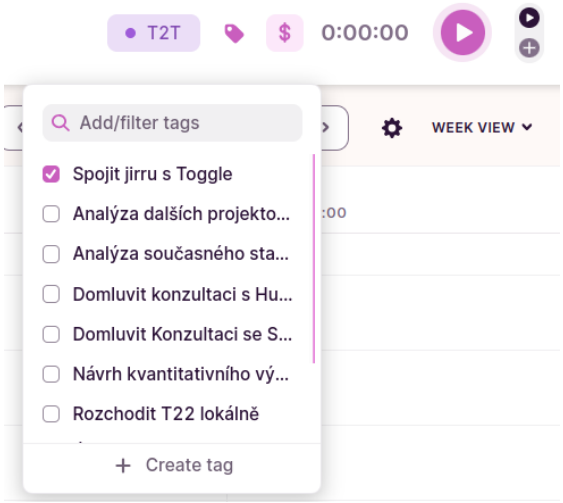
Toggl Track nativně nabízí spojení s projektovým nástrojem Jira [40]. Po zprovoznění synchronizace je uživateli přímo v Jira, u jednotlivých úkolů, nabídnuta možnost zapnout a ukončit časomíru. Uživatel tak nemusí odcházet z projektového nástroje a zapínat stopky v Toggl Track. Data jsou následně synchronizována z Jira směrem do Toggl. Opačným směrem synchronizace neprobíhá. Na každý Toggl workspace je možné napojit pouze jednu instanci Jira.

Při nastavování synchronizace je možné nastavit mapování objektů z Jira na objekty v Toggl Track. Trojici atributů z Jira (Project, Issue, Label) je možné jakkoliv namapovat na Atributy v Toggl Track (Project, Task, Tag, Client). Toto poskytuje velmi velkou flexibilitu ve využívání nástrojů pro uživatele. Ti nejsou omezováni striktními pravidly mapování a mohou proto využívat nástroje tak, jak jim to přijde přirozené a odpovídá jejich pracovní kultuře.

Nevýhody této integrace jsou, že uživatel musí mít prémiový účet Toggl Track, který stojí 20$ měsíčně bez DPH (údaj z 6.10.2023 [2]). Synchronizace navíc probíhá pouze jedním směrem a to z Jira do Toggl. V Jira se tedy neobjeví záznam o tom, jak dlouho uživatel na úkolu pracoval.



Obrázek 2.6 Nastavení spojení mezi Toggl Track a Jira



Obrázek 2.7 Tagy přenesené z Jira do Toggl Track

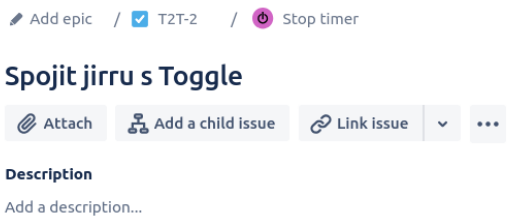
Testování propojení Toggl a Jira

Abych lépe porozuměl této integraci, rozhodl jsem se ji sám na sobě otestovat. Prošel jsem kroky podle návodu [40] a nastavil, aby se mi úkoly z Jira namapovaly na tagy v Toggl. Toto nastavení je vidět na obrázku 2.6. Po konfiguraci integrace proběhla synchronizace s nástrojem Jira, kdy se do Toggl natáhly jako tagy úkoly z Jira, viz obrázek 2.7. V Toggl je nyní možné spustit časomíru nad určitým úkolem bez nutnosti takový úkol v Toggl vytvářet.

Následně jsem zkoušel měřit čas pro různé úkoly. V Toggl jsem pro výběr úkolu využíval tagy, které odpovídaly zadaným úkolům v Jira. V Jira se objevilo nové tlačítko od Toggl Track (ukázka na obrázku 2.8), kterým jsem mohl časomíru zapnout a vypnout přímo u úkolů v Jira. Do Toggl Track se pak časový záznam dostal stejně, jako bych časomíru zapínal tam.

Pozitivní aspekty integrace

Výběr stavů úkolů k synchronizaci – Konfigurace spojení donutí uživatele vybrat si stavy úkolů, které mají být synchronizovány. Například je tak možné definovat, že hotové úkoly už



Obrázek 2.8 Tlačítko Toggl Track v detailu úkolu v nástroji Jira

nemají být synchronizovány do Toggl a uživateli tak při výběru nebudou překážet.

Neomezený počet okamžitých synchronizací – V Toggl Track je možné kdykoliv stisknout tlačítko „synchronizovat“ a nové úkoly z Jira se okamžitě natáhnou do Toggl Track. Oficiální údaj o frekvenci automatické synchronizace se mi nepodařilo dohledat, ale na základě pozorování se jedná nejhůře o jednotky hodin, většinou se záznamy synchronizovaly každou hodinu až dvě.

Možnost mapování projektů, tagů a issues – Při nastavení konfigurace je uživatel donucen vybrat mapování mezi systémy. To sice podle mě není nejlépe uživatelsky zvládnuté, ocenil bych například zvýraznění povinných polí, ale dává to velkou míru flexibility uživatelům v tom, jak chtějí nástroj používat. Mohou tak například nechat tvořit tagy v Toggl podle jednotlivých issues i labels v Jira.

Problémy integrace

Při testování jsem narazil na několik problémů. V této části je chci popsat a následně se z nich poučit a pro Timer2Ticket tyto nedostatky odstranit.

Mizící tlačítko Toggl v Jira – Po zprovoznění synchronizace mi nešla spustit časomíra v Jira. Tlačítko Toggl Track z Jira dokonce na nějakou dobu i zmizelo. Chyba byla v rozšíření Toggl Track pro prohlížeč Chrome, bez kterého synchronizace zjevně nefunguje. Rozšíření se totiž nevypořádalo dobře se založením nového účtu na Toggl Track (který jsem si založil kvůli 30-denní zkušební době na prémiové funkce). Po správném přihlášení do Chrome rozšíření se už v Jira objevilo tlačítko a já mohl začít měřit čas k jednotlivým úkolům.

Chybové hlášky při konfiguraci – Nakonfigurovat správně službu se mi nepodařilo na první pokus, protože informaci o povinných polích ve formuláři jsem dostal až na konci průchodu vícestránkovým konfiguračním formulářem. Pokud je navíc v některé části formuláře chyba, tato část se neuloží a je proto nutné formulář vyplnit celý znovu, včetně správně vyplněných polí. Chybové hlášky, upozorňující na chybu, navíc mizí velmi rychle. Měl jsem dokonce problém ji celou včas přečíst, natož uvažovat nad tím, co tato chyba vůbec znamená. Určení místa, kde se chyba stala také není jednoduché, protože z chybové hlášky toto není jednoznačné, viz obrázek 2.9.

Čas není zaznamenáván v Jira, ale pouze v Toggl – I když je Toggl Track měření času spuštěno přes stopky v nástroji Jira, tak se zaznamenaný čas v Jira neobjeví. Změřený čas je vidět pouze v Toggl Track, kde jsou zpřístupněny reporty uživateli, ale do Jira je tuto informaci nutné přenést ručně.

Obrázek 2.9 Chybová hláška z konfigurace integrace mezi Toggl a Jira

Závěr z testování Toggl Track Jira synchronizace

Zprovoznění existující synchronizace mezi Toggl Track a Jira má své mouchy, ale i tak je relativně jednoduché a intuitivní. Zaznamenávání času do Toggl Track funguje skvěle ať už uživatel zaznamenává přímo v Toggl Track, nebo v Jira. Problém je, že čas strávený na úkolech se mi vůbec nepropisoval do Jira. Je možné, že je k tomu potřeba placená verze Jira, ale nikde jsem o tomto nenašel ani zmínku. Tato integrace tak nesplňuje základní požadavek, který je na nástroj Timer2Ticket kladen. Cílem je mít naměřené časové aktivity přímo v projektovém nástroji.

2.4.2 Průzkum mezi aktuálními uživateli

Vzhledem k tomu, že již existuje řešení na propojení Toggl Track a Jira, rozhodl jsem se udělat průzkum mezi stávajícími uživateli. Cílem bylo především zjistit, jak jsou spokojení s aktuálním řešením, co na aktuální synchronizaci mají a nemají rádi.

Při tvorbě dotazníku jsem se inspiroval radami, jak tvořit dotazník o softwarovém produktu z [41]. Vytvořený dotazník je k dispozici k nahlédnutí v příloze C.

Průzkum jsem prováděl na sociální síti Reddit [42], v Subredditu [1] Jira [43]. Subreddit týkající se Toggl jsem nenašel. V plánu bylo zveřejnění i na diskusních fórech obou společností (Toggl a Atlassian). Toggl ale fórum vůbec nemá a Atlassian má v podmínkách použití zákaz zveřejňovat příspěvky za komerčním použitím [44]. Nástroj Timer2Ticket ale má komerční variantu a předpokládá se, že integrace Toggl Track a Jira bude komerčně využívána, proto jsem se rozhodl dotazník na fórum Atlassian nezveřejňovat.

Dotazník byl tak zveřejněn pouze ve zmiňovaném Subredditu Jira. Příspěvek s odkazem na dotazník [45] jsem zveřejnil ve čtvrtek 19. října 2023. Za 14 dní vidělo tento příspěvek 681 lidí, ale dotazník nevyplnil nikdo. Může to podle mě být z několika důvodů:

- Dotazník byl špatně navržený a uživatelé mu nerozuměli a proto ho nevyplnili
- Uživatelé jsou perfektně spokojeni se stávajícím řešením a nepotřebují další
- Málo lidí využívá spojení Toggl Track a Jira
- Málo takových uživatelů sleduje Jira Subreddit[2].

Ať byl důvod jakýkoliv, dopadl průzkum mezi uživateli fiaskem a já jsem z něj nezískal relevantní data pro tuto práci. Už před zveřejněním dotazníku jsem si byl vědom, že hledám odpovědi v pravděpodobně malé a dosti specifické skupině uživatelů, kterou mám problém cíleně oslovit. Přesto jsem čekával alespoň nějaké odpovědi. Pravděpodobně byl průnik uživatelů synchronizace a čtenářů Jira Subredditu příliš malý.

Tato aplikace není nikde veřejně dostupná, ale nabízí možnost zažádat o účet přes jejich e-mail, uvedený v [46]. Toho jsem využil a o účet pro vyzkoušení této služby jsem zažádal, ale nedostal jsem odpověď.

[1] Subredit označuje zájmovou skupinu, v tomto případě sdružuje lidi, kteří se zajímají o Jira
[2] Jira Subreddit sledovalo ke 2. listopadu 2023 asi 8700 lidí [43]

2.4.3 Integrace Jira a Toggl od společnosti Commutatus

Řešení propojení projektového nástroje Jira a měřiče času Toggl Track vytvořila firma Commutatus [47] pro vlastní účely. Fungování tohoto propojení popisují v článku [46]. Využívají oba zmíněné nástroje a cílem je mít úkoly z Jira nahrané v Toggl Track. Pak vývojář může měřit čas pouze v Toggl Track a čas má zaznamenaný ke správným úkolům, aniž by je musel do Toggl přenášet manuálně. Jejich řešení tedy spočívá pouze v přenesení úkolů z Jira do Toggl.

2.5 Sběr požadavků

V této sekci se věnuji sběru a analýze požadavků na nástroj Timer2Ticket. Nejdříve jsem nastudoval požadavky z předchozích prací, které budu validovat, aktualizovat a upřesňovat podle konzultací se zadavatelem a jinými stakeholdery. Jedná se o kvalitativní průzkum, kdy zjišťuji požadavky od malé skupiny lidí. Výsledky nejsou kvantifikovatelné, ale poskytnou dobrý popis problému.

2.5.1 Rozhovory se stakeholdery

Rozhovory se stakeholdery ze společnosti Jagu s. r. o. [21] jsem si chtěl ověřit aktuálnost a správnost požadavků sesbíraných v rámci předchozích prací na téma Timer2Ticket [1, 4, 5]. Zároveň jsem chtěl dosáhnout upřesnění požadavků, které mají být implementovány v rámci této práce a případně odhalit nové.

V rámci přípravy na rozhovory jsem prostudoval požadavky z předchozích prací Víta Štefana [1], Jakuba Čermáka [4] a Martina Paula [5]. Následně jsem připravil semi-struktorovaný rozhovor, ve kterém jsem cílil na aktualizaci a upřesnění těchto požadavků.

Před rozhovorem jsem si řádně prostudoval dokumentace Jira [22] a Redmine [3]. Při přípravě jsem se inspiroval [48, 49, 50]. Rozhovor jsem připravil tak, abych v úvodu zjistil, jaké jsou cíle projektu a kdo bude zákazníkem výsledného produktu. Dále jsem se snažil zjistit, co uživatele trápí na stávajících řešeních a jak by si představovali fungování výsledného nástroje. U jednotlivých požadavků jsem se snažil jít více do hloubky a ptát se na detailní představy stakeholderů o fungování aplikace. Na konci rozhovoru jsem si s respondentem zvalidoval, jestli jsem ho správně pochopil a domluvili jsme se na dalším postupu.

V průběhu rozhovoru jsem si dělal zápisky do připravené osnovy, kterou lze najít v příloze A. Jednotlivé zápisky jsem následně shrnul do následujících odstavců. Rozhovory spolu s analýzou požadavků z předchozích prací nakonec posloužily jako vstupy pro sepsání funkčních a nefunkčních požadavků.

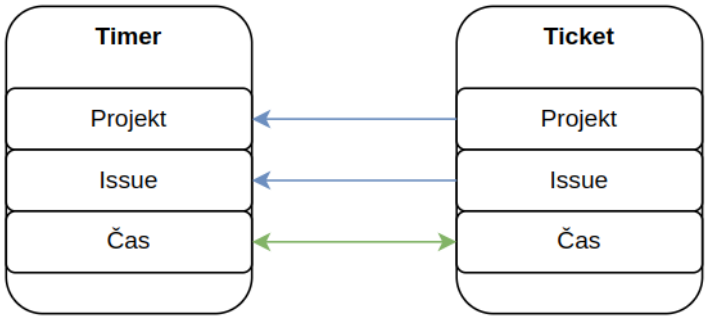
Jiří Hunka

Jiří Hunka je jednatel a většinový majitel [51] společnosti Jagu s. r. o. [21] a vedoucí této diplomové práce. K diskusi nad požadavky jsme se sešli 18. října 2023. Získal jsem jeho požadavky na aplikaci z pohledu majitele firmy a osoby, která bude produkt využívat jak pro interní potřeby firmy, tak ho komerčně nabídne i dalším osobám.

Synchronizace mezi měřičem času Toggl a projektovým nástrojem Jira má probíhat stejně a jsou na ni kladeny stejné požadavky, jako tomu bylo v úvodní práci Víta Štefana [1]. Podle Jiřího Hunky funguje synchronizace správně a využívá ji ve svých projektech v rámci SP1 a SP2 na FIT ČVUT [52] i ve firmě Jagu s. r. o. Upozornil mě ale na jinou logiku v Jira ohledně projektů a štítků. Bude podle něj třeba najít rozumné mapování tak, aby přišlo intuitivní stávajícím uživatelům Timer2Ticket, kteří využívají Redmine, i novým uživatelům využívající Jira.

Cílem synchronizace časů mezi Redmine a Jira je, aby měl odběratel [3] (využívající Jira)

[3] Odběrateletem zde rozumíme zaměstnavatele vývojáře, nebo zákazníka SW firmy



Obrázek 2.10 Směry synchronizace objektů mezi Timer a Ticket

v reálném čase informaci o tom, kolik času bylo na jednotlivých úkolech odpracováno. Zároveň bude pro vývojáře jednodušší využívat pouze jeden projektový nástroj a nechat časy synchronizovat automaticky, protože manuální přenášení stojí čas a dějí se při něm chyby. Pro oba projektové nástroje navíc existují doplňky a firmy v nich mají zanesené vlastní workflow. Synchronizace jim umožní tyto doplňky využívat a nebude je nutit přecházet na jiný projekt do jiného projektového nástroje.

Využívání Timer2Ticket by mělo chránit dodavatele [4] i odběratele a pomoci tak předejít překvapením na konci měsíce z množství vypotřebovaného času. Jiří Hunka předpokládá, že tuto synchronizaci budou využívat jak samostatní vývojáři, tak společnost Jagu s. r. o.

Jiří Hunka nevidí zásadní rozdíl mezi fungováním synchronizace Toggl-Redmine a Redmine-Jira. Podle něj by uživatel měl při nastavování zvolit, který systém je „Timer“ a který „Ticket“. Logika by tak byla stejná, jako je tomu nyní mezi Toggl Track a Redmine. Toto je schématicky znázorněno na obrázku 2.10. Modré šipky znázorňují jednosměrnou synchronizaci směrem z ticketovacího nástroje do nástroje na měření času a oboustranná šipka značí, že zadaný čas se synchronizuje mezi nástroji oběma směry.

Dalším požadavkem je, aby bylo umožněno mapování 1:N mezi úkoly. Zákazník totiž může vytvořit jeden úkol v Jira, požadující nějakou komplexní funkcionalitu. V interním Redmine pak ale může být onen úkol rozložen do menších částí a čas bude vykazován k jednotlivým podúkolům. V Jira by pak měl být vidět celkový vykázaný čas na všech podčástech tohoto úkolu. To zapadá do konceptu z obrázku 2.10. Protože už nyní mohu mít více různých časových záznamů z Toggl navázaných na jeden tag, tím pádem na jeden úkol v Redmine. Podobná logika je možná i mezi Redmine a Jira.

Oldřich Malec

Oldřich Malec je projektovým manažerem a menšinovým vlastníkem [51] společnosti Jagu s. r. o. Rozhovor se uskutečnil 10. října 2023 online. Soustředil jsem se v něm především na přidání projektového nástroje Jira. Požadavky jsme rozdělili na 2 skupiny, podle cílového zákazníka.

Základní funkce aplikace Timer2Ticket (synchronizace časových záznamů mezi měřičem času a projektovým nástrojem) je podle Oldřicha Malce primárně určena pro vývojáře, kterým má za cíl usnadnit vykazování stráveného času na jednotlivých úkolech a projektech. V tuto chvíli je tímto způsobem implementovaná integrace mezi Toggl Track a Redmine. Přidání nového projektového nástroje sebou nenese další nové požadavky oproti původní integraci.

Přidáním projektového nástroje Jira očekává Oldřich Malec větší množství problémů se synchronizací časových aktivit, protože původní návrh Víta Štefana [1] počítal především s nástrojem Redmine, ale Jira se v některých atributech výrazně liší. Z povahy nástroje by měl uživatel chodit

[4] Dodavatelem zde rozumíme zaměstnance, nebo SW firmu

do webového rozhraní co nejméně, a proto je nutné najít jiný způsob komunikace a umět dát uživateli vědět, když se něco pokazí. Pravděpodobně e-mailem.

Druhá větev požadavků se týká synchronizace mezi dvěma projektovými nástroji. V tomto případě je primárním zákazníkem společnost Jagu s. r. o., která má zákazníky, kteří pracují s Jira a chtějí v ní mít i výkaz práce. Primárním projektovým nástrojem Jagu s. r. o. je ale Redmine. Bude třeba nalézt mapování mezi projektovými systémy Jira a Redmine a synchronizovat časové záznamy mezi nimi. Úkolem Timer2Ticket podle něj není přenášet mezi systémy jednotlivé požadavky, ale pouze časové záznamy. Timer2Ticket tedy bude počítat s tím, že požadavky existují v obou systémech, pouze hledá vhodné mapování a řeší nekonzistence.

Pro synchronizaci mezi projektovými nástroji by si Oldřich Malec představoval nové *Enterpise* předplatné nad rámec aktuálních (Hobby, Junior, Senior).

Sára Sovičková

Sára Sovičková je studentkou FIT, která na projektu „Chytrá domácnost“ využívá nástroj Timer2Ticket. Dále vypomáhá ve společnosti Jagu s. r. o. na projektu se zákazníkem, který vyžaduje vykazování času v projektovém nástroji Jira. Sešli jsme se k rozhovoru 18. října 2023 a ptal jsem se primárně na aktuální problémy přenášení dat mezi zákazníkovým projektovým nástrojem Jira a interním Redmine.

Aktuální flow je takové, že issue od zákazníka je zadán v Jira. Sára Sovičková, jakožto projektový manažer, založí stejný požadavek v Redmine a do Jira vloží odkaz na tento nový issue. Čas strávený na požadavku se následně vykazuje v Redmine a v Jira se tento čas nedá vyčíst, protože přenášení času by bylo zbytečně administrativně náročné a zákazník se může na strávený čas podívat přes odkaz do Redmine.

Výrazně by jí zjednodušilo práci automatické přenášení úkolů z Jira do Redmine, i přenášení stráveného času z Redmine do Jira. Myslí si, že by se o tuto integraci měl starat pouze projektový manažer a ostatní členové týmu by pak ani nepotřebovali vědět, že je Jira jakýmkoliv způsobem do procesu zapojená.

Dotkli jsme se také problému, jak vykazovat v Jira čas strávený na projektu, který není vykázaný ke konkrétnímu úkolu. Jsou to například administrativní aktivity, které nemají vlastní úkol. Jira totiž takové vykazování času neumožňuje, proto bude pravděpodobně třeba zřídit v Jira „fallback“ (výchozí) úkol, kam se budou takové časy synchronizovat.

Velmi podobný problém bude pravděpodobně vznikat při vykazování jednotlivými členy týmu, protože ne všichni mají založený účet v Jira. Bylo by tedy vhodné zvolit výchozího uživatele, za kterého bude vykázán čas v Jira v případě, že nástroj Timer2Ticket nedokáže najít Redmine uživateli jeho Jira účet. Vznikne tak potenciální nekonzistence v tom, kdo jak dlouho pracuje na projektu, ale bude správně alespoň suma vykázaného času.

Frekvence synchronizace mezi nástroji by podle Sáry Sovičkové měla být nastavitelná, některý zákazník vyžaduje přístup k informaci o stráveném čase nad projektem častěji (synchronizace dvakrát denně je podle ní dostatečná), jinému zákazníkovi bude údajně stačit synchronizace jednou týdně.

Jakub Čermák

Student FIT a autor nového frontendu Timer2Ticket z bakalářské práce [4]. Pro společnost Jagu s. r. o. dále analyzoval požadavky a možnosti nástroje Jira a Redmine podle specifických potřeb firmy [53, 54]. Na schůzce 19. října 2023 mi potvrdil aktuální pracovní flow tak, jak mi ho popsala Sára Sovičková a seznámil mě se svou analýzou synchronizace Redmine a Jira.

Podle něj je cílem mít v obou systémech (Redmine a Jira) stejná a aktuální data. Znamená to, že synchronizace musí brát v potaz poslední aktualizaci úkolů a propsat tyto změny do druhého systému. Není tedy jeden primární a druhý sekundární, ale za pravdivý údaj považujeme ten nejnovější. Kromě času bude také třeba synchronizovat další atributy, konkrétně: název úkolu, frontu (typ), stav, přiřazenou osobu, prioritu a pole odkazující na merge request.

Bylo by ideální mít správně synchronizované uživatele podle jejich výkazu ve druhém systému. Toto řešení bude ale pravděpodobně velmi náchylné na chyby, proto bude podle něj stačit vykazovat čas do Jira přes jednoho uživatele.

Redmine taktéž, na rozdíl od Jira, nabízí možnost zaznamenat čas i k projektu bez přiřazeného úkolu. Pro takové případy by v Jira měl vzniknout „fallback“ úkol, do které by se propsal čas vykázaný k projektu.

2.6 Funkční a nefunkční požadavky

Na základě studia předchozích závěrečných prací na téma Timer2Ticket a rozhovorů se stakeholdery vzešly následující funkční a nefunkční požadavky. Jejich kompletní seznam včetně výsledného stavu implementace je k dispozici v příloze D. Požadavky jsou opatřeny prioritou a odhadem náročnosti.

2.6.1 Prioritizace požadavků

Pro prioritizaci požadavků jsem zvolil metodu MoSCoW [55], která rozděluje požadvky podle priorit. *Must have* požadavky jsou takové, bez kterých nemá produkt smysl a proto je nutné je implementovat. Bez *Should have* požadavků lze produkt vypustit na trh, ale jejich absence bude uživatelům způsobovat nemalé problémy v používání produktu. Jako *Could have* jsou označeny požadavky, které by bylo dobré mít ve výsledné implementaci, ale jejich absence nezpůsobí problémy pro uživatele. Poslední *Won't have* požadavky jsou takové, které nemají být vůbec implementovány a je to takto explicitně označeno.

Priority jednotlivým požadavkům byly přiřazeny po konzultaci s Jiřím Hunkou, vedoucím této práce. Seřazením požadavků podle priorit vzniká plán pořadí návrhu implementace jednotlivých požadavků. Při implementaci začnu s *Must have* požadavky a budu pokračovat sestupně podle priority.

2.6.2 Kategorizace požadavků

Kromě základního členění na *Funkční* a *Nefunkční* požadavky [56] využiji ještě kategorizaci FURPS (functionality, usability, reliability, performance, supportability) [57]. Podle této metody lze požadavky rozdělit do pěti skupin. *Functionality* označuje požadavky na fungování aplikace. *Usability* označuje požadavky, které přispívají použitelnosti a zlepšují uživatelův zážitek. Požadavky ze skupiny *Reliability* se zaměřují na spolehlivost, *Performance* na výkon a kategorie *Supportability* nás nutí myslet i na budoucí vývoj a rozšíření aplikace.

2.6.3 Požadavky na synchronizaci nástrojů Toggl Track a Jira (TTJ)

Synchronizace nástroje pro měření času Toggl Track a projektového nástroje Jira je určená především pro vývojáře, kterým by měla usnadnit vykazování času pro zaměstnavatele nebo zákazníka. Cílem integrace Jira do nástroje Timer2Ticket je umožnění vykazovat naměřený čas do více projektových nástrojů.

TTJ01 – Přidání projektového nástroje Jira do aplikace Timer2Ticket

Nástroj Timer2Ticket bude rozšířen o projektový nástroj Jira. Bude možné nastavit spojení mezi nástrojem Toggl Track a Jira pro synchronizaci časových záznamů. Směrem z Jira do Toggl

Track budou synchronizovány projekty a úkoly, obousměrně pak budou synchronizovány časové záznamy.

Náročnost : Střední

Priorita : Must have

Dotčené části : Frontend, API, core

Typ požadavku : Funkční – functionality

TTJ02 – Konfigurace spojení

Uživatel bude mít možnost nakonfigurovat synchronizaci projektového nástroje Jira a měřiče času Toggl Track. Tato konfigurace bude možná z webového rozhraní Timer2Ticket podobně jako je nyní konfigurováno spojení mezi Toggl Track a Redmine. Ke konfiguraci bude webový klient vyžadovat minimální možné množství údajů od uživatele. Implementace bude vycházet z návrhu Jakuba Čermáka [4].

Náročnost : Střední

Priorita : Must have

Dotčené části : Frontend

Typ požadavku : Funkční – usability

TTJ03 – Synchronizace úkolů a projektů

Nástroj Timer2Ticket zajistí jednosměrnou synchronizaci úkolů (issues) a projektů z projektového nástroje Jira do nástroje Toggl Track. Úkoly z Jira budou mapovány do Toggl Track ve formě tagů. Pro každý projekt v Jira bude vytvořen nový projekt v Toggl Track. Synchronizovány budou úkoly pouze s odpovídajícím stavem (viz požadavek TTJ07 2.6.3).

Náročnost : Střední

Priorita : Must have

Dotčené části : API, Core

Typ požadavku : Funkční – functionality

TTJ04 – Synchronizace časových záznamů

Nástroj Timer2Ticket zajistí obousměrnou synchronizaci časových záznamů mezi projektovým nástrojem Jira a nástrojem pro času Toggl Track. Pokud se v jednom z nástrojů objeví nový časový záznam, automaticky se při příští synchronizaci propíše do druhého nástroje. Stejně tak bude Timer2Ticket propisovat změny a smazání časového záznamu do druhého systému. Podrobné fungování synchronizace časových záznamů je popsáno v kapitole 2.6 diplomové práce Víta Štefana [1].

Náročnost : Střední

Priorita : Must have

Dotčené části : Frontend, Core

Typ požadavku : Funkční – functionality

TTJ05 – Synchronizace času bez vybraného úkolu

V nástroji Toggl Track je možné spustit časomíru tak, že čas bude měřen pouze k projektu (bez vybraného úkolu). V takovém případě je také třeba zajistit synchronizaci času mezi Toggl Track a Jira. Uživatel dostane při konfiguraci spojení na výběr, jak se k takovému časovému záznamu má Timer2Ticket chovat.

Pokud bude nejdřív čas v Toggl Track změřen bez úkolu, ale později mu bude tag v Toggl Track přiřazen, musí se tato změna propsat i do Jira.

Náročnost : Střední

Priorita : Must have

Dotčené části : Core

Typ požadavku : Funkční – functionality

TTJ06 – Výběr výchozího úkolu

Uživatel bude mít možnost při konfiguraci spojení vybrat z existujících úkolů v Jira per projekt takový, do kterého se budou vkládat časy bez přiřazeného úkolu. Pokud uživatel úkol nevybere, bude v Jira vytvořen nový „Fallback“ úkol pro tyto účely. Do něj budou spadat všechny časové záznamy bez přiřazeného úkolu.

Pokud bude v Toggl Track tag úkolu k časovému záznamu přidán, musí tento Časový záznam být přesunut z „Fallback“ úkolu do úkolu nového.

Náročnost : Střední

Priorita : Should have

Dotčené části : Frontend, API, Core

Typ požadavku : Funkční – functionality

TTJ07 – Filtrace úkolů k synchronizaci

V nastavení spojení bude mít uživatel možnost filtrovat mezi úkoly v Jira a omezit tak množinu vytvořených tagů v Toggl Track. Uživatel tak bude mít například možnost omezit synchronizaci tagů pouze na úkoly v určitém stavu (TO DO, In Progress). Tagy, které přestanou odpovídat filtru (v Jira se změní jejich stav) budou odebrány z množiny nabízených tagů v Toggl Track. Například si tak uživatel bude moct odstranit z nabídky tagů úkoly, které jsou již hotové. Bude tak mít méně tagů, ze kterých bude vybírat při spouštění časomíry.

Podobnou funkcionalitu nabízí nativní integrace těchto dvou nástrojů od Toggl [2, 40], kterou jsem zkoumal v kapitole 2.4. Navíc dokonce nabízí více možností mapování objektů mezi nástroji. Například je tak možné namapovat projekty do tagů. To už ale považuji za velmi *Nice To Have* funkcionalitu a s implementací nastavení mapování zatím nepočítám. Zároveň se nabízí přidání filtrace požadavků i pro stávající služby (Redmine).

Náročnost : Střední

Priorita : Could have

Dotčené části : Frontend, API, Core

Typ požadavku : Funkční – usability

Stejná filtrace bude implementována také pro konfiguraci projektového nástroje Redmine, případně pro další projektové nástroje implementované později. Chování bude co nejpodobnější napříč konfiguracemi. Filtraci pro konfiguraci nástrojů pro měření času (Toggl Track) nemá smysl dělat, protože z nich nejsou úkoly synchronizovány jinam.

TTJ08 – Hlášení chyb

Timer2Ticket je nástroj, do kterého by měl uživatel chodit co nejméně. Je proto důležité, aby se o případné chybě v synchronizaci (chybějící objekty, nedostatečná přístupová práva, ...) mohl dozvědět i jinak, než přímo v aplikaci. Timer2Ticket tedy bude disponovat možností notifikace uživatele i jiným způsobem (e-mail, Sentry.io [58]). Způsob a frekvenci notifikací bude mít uživatel možnost nastavit ve webovém rozhraní Timer2Ticket.

Náročnost : Střední

Priorita : Should have

Dotčené části : Frontend, API, Core

Typ požadavku : Funkční – usability

2.6.4 Požadavky na synchronizaci projektových nástrojů (SPN)

V této části se věnuji požadavkům na synchronizaci časových záznamů mezi projektovými nástroji Jira a Redmine. Požadavky je ale možné abstrahovat od konkrétních nástrojů a je možné je využít i pro přidání dalšího projektového nástroje do Timer2Ticket. Synchronizace projektových nástrojů zatím v Timer2Ticket není implementována, protože Timer2Ticket zatím fungoval pouze s jedním projektovým nástrojem (Redmine).

SPN01 – Zachovat princip Timer a Ticket nástroje

Při propojování dvou projektových nástrojů bude nutné vybrat, který nástroj je typu „Timer“ (nástroj pro měření času) a který typu „Ticket“ (projektový nástroj). Synchronizace bude následně probíhat podle stejného schématu, jako je znázorněno na obrázku 2.10. Z „Ticket“ budou jednosměrně do „Timer“ synchronizovány projekty a úkoly. Synchronizace časových záznamů bude probíhat obousměrně podle stejného principu, který je nyní využit mezi Toggl Track a Redmine (kapitola 2.6 [1]).

Náročnost : Nízká

Priorita : Must have

Dotčené části : Frontend, API, Core

Typ požadavku : Funkční – usability, Nefunkční – supportability

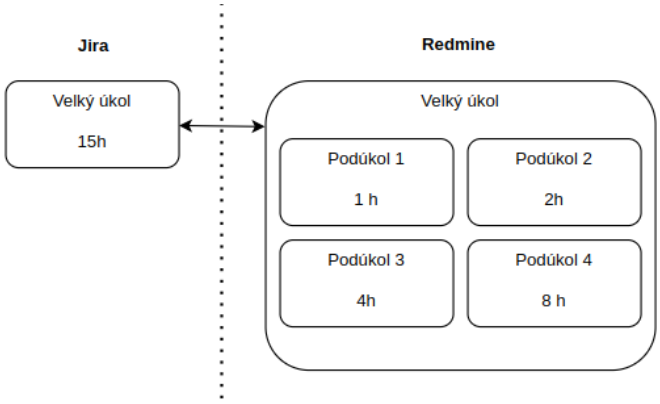
Projektové nástroje bude ve spojení možné využít jak jako „Ticket“, tak i „Timer“ nástroj. Nástroje pro měření času budou moct ve spojení figurovat pouze jako „Timer“. Bude tedy možné spojit dva projektové nástroje, z nichž jeden bude „pouze“ v roli „Timer“, ale už nebude možné spojit dva nástroje pro měření času. Takové spojení nedává z uživatelského pohledu smysl.

SPN02 – Spojení mezi Redmine a Jira

Do Timer2Ticket přibude možnost spojení mezi projektovými nástroji Jira a Redmine. Oba tyto nástroje bude možné využít jako nástroj typu „Timer“ i „Ticket“. Synchronizace času, projektů a úkolů pak bude probíhat podle původního schématu (kapitola 2.6 [4]).

Náročnost : Vysoká

Priorita : Must have



Obrázek 2.11 Mapování úkolu mezi projektovými nástroji 1:N, bez notace, vytvořeno pomocí [59]

Dotčené části : Frontend, API, Core

Typ požadavku : Funkční – functionality

SPN03 – Přizpůsobení webového rozhraní pro spojení projektových nástrojů

Nyní uživatel při konfiguraci spojení vybírá mezi „primárním“ a „sekundárním“ nástrojem. Implementací dalšího projektového nástroje do Timer2Ticket může vzniknout i spojení mezi projektovými nástroji. Proto dojde k úpravě webového rozhraní a spojení bude konfigurováno mezi nástrojem typu „Timer“ a nástrojem typu „Ticket“.

Náročnost : Nízká

Priorita : Must have

Dotčené části : Frontend

Typ požadavku : Funkční – usability

SPN04 – Mapování úkolů 1:N

Bude umožněno vykázat čas k jednomu úkolu z nástroje „Ticket“ z více úkolů v nástroji typu „Timer“. Například pokud zákazník zadá ve své Jira nějaký požadavek jako jeden úkol, bude možné tento úkol rozložit na podúkoly v interním Redmine a součet časů na těchto podúkolech úkolech bude následně vykázán v původním úkolu v Jira, viz obrázek 2.11. Toto rozložení a složení úkolu s mapováním na jeden úkol ve druhém nástroji bude možné v Redmine i Jira.

Náročnost : Vysoká

Priorita : Must have

Dotčené části : API, Core

Typ požadavku : Funkční – functionality

2.6.5 Hlášení chyb uživateli (NOT)

Zapojením dalších nástrojů do Timer2Ticket očekávám větší míru chybovosti v synchronizaci. Obzvláště chybějící objekty při synchronizaci mezi projektovými nástroji nebude možné vždy vyřešit automaticky. Povaha nástroje Timer2Ticket je taková, že by do něj měl uživatel chodit co nejméně, proto není efektivní informovat ho o chybách pouze ve webové aplikaci. Tím bychom uživatele nutili kontrolovat pravidelně synchronizace a trochu tak popírali prvotní záměr Timer2Ticket. Je proto třeba vybrat jiný nástroj, který doručí zprávu uživateli bez toho, aby musel aktivně chybu vyhledávat a otevírat webového klienta.

NOT01 – E-mailové notifikace

Nástroj Timer2Ticket umožní uživatelům konfiguraci e-mailových notifikací. Uživatelé si budou moct nastavit frekvenci a seskupování notifikací (například jednou denně).

Náročnost : Střední

Priorita : Could have

Dotčené části : Core, API, Frontend

Typ požadavku : Funkční – usability

NOT02 – Hlášení chyb do Sentry.io

Sentry.io [58] je nástroj pro zaznamenávání a zpracování chyb. Umožňuje uživatelům odchytávat velké množství chyb z běžících aplikací a dále s nimi pracovat. Vzhledem k cílové skupině zákazníků (vývojáři) je Sentry.io řešení, které pravděpodobně ze své praxe pravděpodobně znají. Zaznamenávání chyb do Sentry.io by také delegovalo jakoukoliv práci se seskupováním a tříděním chyb z nástroje Timer2Ticket právě na Sentry.io. Uživatelé by také dostali do rukou lepší nástroj na sledování chyb, než jsou e-mailové notifikace. Nevýhodou je pak nutnost znalosti jiného nástroje.

Timer2Ticket bude chyby zasílat do nástroje Sentry.io. Uživatelům umožní ve webovém klientu konfiguraci tohoto nástroje.

Náročnost : Střední

Priorita : Should have

Dotčené části : Core, API, Frontend

Typ požadavku : Funkční – usability

2.6.6 Podpora webhooks (PWH)

Zapojení webhooks má potenciál zrychlit synchronizaci mezi nástroji, které webhooks podporují (Toggl Track [60], Jira[61]). U ostatních nástrojů by vše zůstalo při starém a synchronizace by se prováděla přes API v definované časy.

PWH01 – Konfigurace webhooks

Uživateli s členstvím podporující webhooks bude umožněna jejich konfigurace pro nástroje, kde je to možné. Synchronizace nástrojů pak bude moct probíhat v podstatě okamžitě.

Náročnost : Nízká

Priorita : Could have

Dotčené části : Core, API, Frontend

Typ požadavku : Funkční – usability

PWH02 – Modifikace členství

Implementací webhooks do nástroje Timer2Ticket vzniká narušení stávajícího systému členství. To je totiž nastavené tak, aby docházelo k synchronizaci v pravidelných intervalech. Přidáním webhooks se synchronizace stane okamžitou a bude třeba na to reagovat úpravou členství.

Náročnost : Nízká

Priorita : Could have

Dotčené části : Core, API, Frontend

Typ požadavku : Funkční – usability

PWH03 – Podpora webhooks v Toggl Track

Ve webovém klientu bude uživatelům s příslušným členstvím umožněno zprovoznění webhooks pro Toggl Track. Ostatní části aplikace pak budou schopny webhooks zpracovat.

Náročnost : Střední

Priorita : Could have

Dotčené části : Core, API, Frontend

Typ požadavku : Funkční – usability

PWH04 – Podpora webhooks v Jira

Ve webovém klientu bude uživatelům s příslušným členstvím umožněno zprovoznění webhooks pro Jira. Ostatní části aplikace pak budou schopny webhooks zpracovat.

Náročnost : Střední

Priorita : Could have

Dotčené části : Core, API, Frontend

Typ požadavku : Funkční – usability

2.6.7 Požadavky na Enterprise verzi Timer2Ticket (ENT)

Enterprise verze Timer2Ticket bude synchronizovat úkoly v rámci projektu mezi dvěma projektovými nástroji Redmine a Jira. Níže specifikované požadavky je možné abstrahovat a využít i mezi jinými projektovými nástroji, ale vzhledem k unikátnosti požadavků zadavatele se mi to nejeví jako příliš pravděpodobné. Následující požadavky vychází především z analýzy Jakuba Čermáka pro společnost Jagu s. r. o [53, 54].

Hlavním cílem tohoto spojení je, aby bylo umožněno týmu, který je zvyklý spravovat svůj projekt v Redmine, lépe fungovat s klientem, který má zavedený projektový nástroj Jira. V Jira jsou vývojovému týmu zadávány úkoly, které je třeba přenést do Redmine. V Redmine zase probíhá většina změn úkolů (priorita, stav, atd.). Tyto změny je zase nutné propsat do Jira. Je

ovšem potřeba také kontrolovat a uvažovat změny ticketu v Jira a udržet stejný a konzistentní stav mezi systémy.

Pro „Enterprise“ požadavky vynechávám informaci o dotčených částech systému, protože nakonec nebudou v rámci Timer2Ticket implementovány. Nahrazuji to ale informací odhadu rizika implementace. Riziko odhaduji podle množství věcí, které implementují oba systémy (Redmine a Jira) jinak. Čím více se liší přístup, tím větší rizko, že implementovat požadavek bude nereálné, nebo příliš složité.

ENT01 – Nastavení synchronizace jedním uživatelem

Projektový manažer (nebo jiný uživatel) s dostatečnými právy v obou systémech bude mít možnost zprovoznit spojení na vybraném projektu. Timer2Ticket „Enterprise“ bude následně obousměrně aktualizovat systémy a udržovat je v aktuálním a konzistentním stavu.

Náročnost : Střední

Riziko : Střední

Priorita : Won’t have

Typ požadavku : Funkční – functionality, usability

ENT02 – Vytvoření úkolu

Pokud vznikne v Jira nový úkol, bude mu vytvořen „dvojník“ v Redmine. Jira je jediným zdrojem pravdy pro nově vytvářené úkoly. Úkoly vytvořené v Redmine nebudou přenášeny do Jira. V Jira ale vznikne, nebo bude při konfiguraci vybrán uživatelem „Fallback“ úkol pro čas zaznamenaný k úkolu, který existuje v Redmine, ale neexistuje v Jira.

Náročnost : Střední

Riziko : Nízké

Priorita : Won’t have

Typ požadavku : Funkční – functionality

ENT03 – Synchronizace atributů úkolů mezi nástroji

Aplikace bude synchronizovat vybrané atributy úkolů mezi projektovými nástroji Jira a Redmine. Nebude ovšem synchronizovat všechny atributy, ale pouze vybranou podmnožinu z nich:

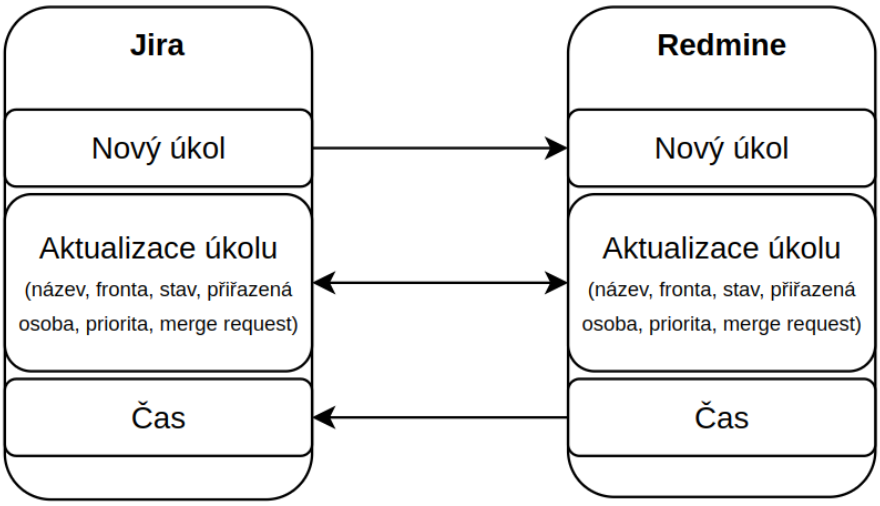
Název úkolu

Odkaz na úkol ve druhém nástroji – Oba nástroje umožňují definici custom fields, to bude využito pro uložení odkazu.

Fronta (Tracker) – Fronta (Tracker) určuje v Redmine typ požadavku (chyba, požadavek, atd.). V Jira tomu odpovídá atribut „type“.

Přiřazená osoba – Přiřazená osoba se může v průběhu života požadavku měnit a je třeba toto propagovat do druhého nástroje. Ne všichni uživatelé ale mají účet v obou systémech. V případě, že nebude nalezen uživatel ve druhém systému, dostane v něm úkol přiřazený „Fallback“ uživatel.

Stav úkolu – Před zprovozněním synchronizace bude třeba dbát na to, aby přípustné stavy úkolů byly v obou nástrojích stejné. Pokud toto nebude zajištěno, bude docházet k chybám a stavy nebudou správně synchronizovány.



Obrázek 2.12 Schéma synchronizace objektů mezi Jira a Redmine v Enterprise verzi, bez notace, vytvořeno pomocí [59]

Priorita – Před zprovozněním synchronizace bude třeba dbát na to, aby přípustné priority byly v obou nástrojích stejné. Pokud toto nebude zajištěno, bude docházet k chybám a priority nebudou správně synchronizovány.

Pole odkazující na merge request – Pro lidsky přívětivější provázanost budou mít úkoly v obou systémech custom field pro odkaz na úkol do druhého systému. Toto pole lze navíc využít pro strojovou kontrolu, zda již tento úkol má svého „dvojníka“ ve druhém systému.

Pokud dojde ke změně v některém z výše uvedených atributů, bude změna propsána do druhého nástroje. V případě kolizí bude za pravdivý záznam považován ten novější. Schématicky jsou směry synchronizace znázorněny na obrázku 2.12.

Náročnost : Extrémní

Riziko : Vysoké

Priorita : Won’t have

Typ požadavku : Funkční – functionality

ENT04 – Synchronizace časových záznamů

Zdrojem pravdy pro strávený čas na úkolu bude nástroj Redmine. Pro zjednodušení vykazování času a předcházení problémům s mapováním uživatelů bude do Jira přenášena pouze suma časů za celý den. V Jira tak vznikne výkaz za celou společnost (bez uživatelů) po jednotlivých dnech.

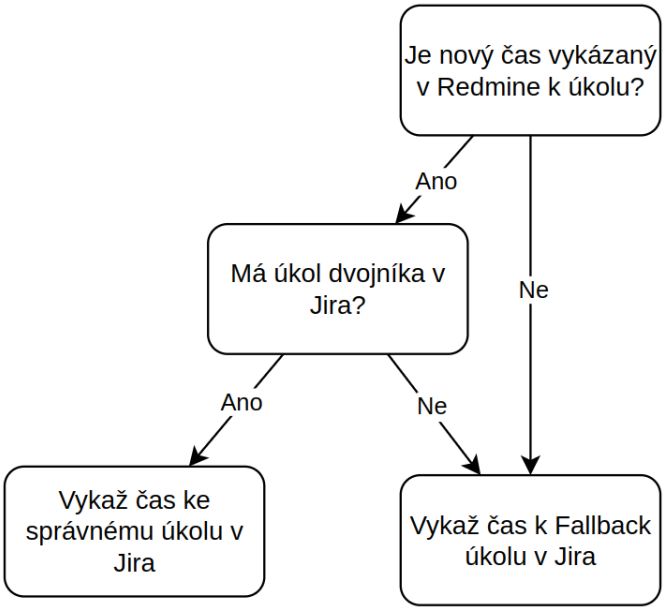
Pro vykazování času k úkolům, které existují v Redmine, ale neexistují v Jira, vznikne v Jira „Fallback“ úkol, do kterého budou vykazovány časy vykázané k jiným, nebo žádným úkolům v Redmine. Schématicky je toto znázorněno na obrázku 2.13.

Náročnost : Střední

Riziko : Střední

Priorita : Won’t have

Typ požadavku : Funkční – functionality



Obrázek 2.13 Rozhodovací strom pro synchronizaci časů v Enterprise verzi, bez notace, vytvořeno pomocí [59]

ENT05 – Mapování uživatelů

Uživatelé, kteří mají účet v obou systémech budou mapováni na sebe a budou na ně při synchronizaci přiřazovány úkoly podle aktuální změny. Pokud některý z uživatelů nebude mít účet v obou systémech, budou jeho úkoly přiřazovány na zvoleného „Fallback“ uživatele.

Náročnost : Střední

Riziko : Nízké

Priorita : Won't have

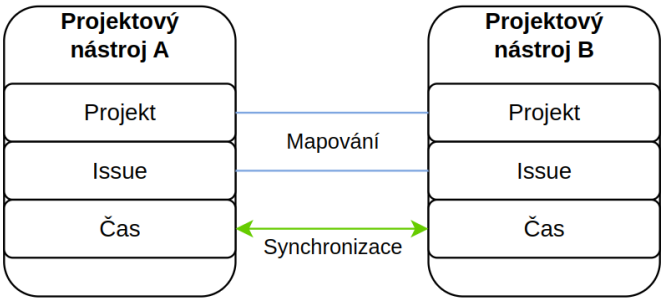
Typ požadavku : Funkční – functionality

2.6.8 Doplnění požadavků na synchronizaci projektových nástrojů

Ve čtvrtek 9. listopadu 2023 jsem se znovu sešel k diskusi nad požadavky s Jiřím Hunkou a Oldřichem Malcem. Dohodli jsme se, že „Enterprise“ verzi Timer2Ticket není třeba dál rozebírat, a že tyto funkcionality nemají být součástí nástroje Timer2Ticket. Od Timer2Ticket je totiž očekávána pouze synchronizace časových záznamů a synchronizací celých projektů bychom uživatelům ztížili orientaci v nástroji a implementovali něco, o co není zájem. Pro takovouto synchronizaci by měl sloužit jiný nástroj.

Dále jsme na schůzce odhalili, že jsem nesprávně pochopil požadavky na synchronizaci projektových nástrojů, která součástí Timer2Ticket být má. Proto vznikla tato část, která doplňuje a upřesňuje požadavky z kapitoly 2.6.4.

Hlavní chyba, kterou jsem v předchozí analýze udělal byla, že v projektových nástrojích nemají být automaticky zakládány při synchronizaci úkoly. Bude tam tedy muset fungovat trochu jiná logika, než pro synchronizaci nástroje pro měření času a projektovým nástrojem. Pro



Obrázek 2.14 Schéma synchronizace mezi projektovými nástroji, bez notace, vytvořeno pomocí [59]

čas, který bude naměřen v Toggl Track k projektu (při spojení s Jira) pak bude třeba navrhnout, kam záznam patří. Přikládám aktualizovaný obrázek komunikace mezi systémy (původní schéma 2.10). Správně bude komunikace probíhat podle schématu 2.14. Projekty a úkoly budou zakládány pouze v nástrojích, které jsou skutečnými měřiči času, nikoliv v projektových nástrojích. Modré čáry spojující projekty a úkoly označují pouze mapování mezi systémy. Časové záznamy budou synchronizovány mezi systémy oběma směry.

Úpravou tohoto požadavku efektivně ztrácí smysl požadavek SPN01 2.6.4 o zachování principu nástrojů Timer a Ticket. Při spojení dvou projektových nástrojů si budou efektivně tyto nástroje rovny a nebude tak záležet na tom, který z nich je primární a který sekundární nástroj.

SPN01B – Vytváření objektů v projektových nástrojích

Tento požadavek nahrazuje SPN01 2.6.4, který mluvil o zachování principu nástrojů „Timer“ a „Ticket“ i pro projektové nástroje.

Timer2Ticket umožní uživateli při spojení dvou projektových nástrojů nastavit mapování projektů a úkolů. Ani v jednom ze systémů nebudou zakládány nové úkoly, Timer2Ticket bude pouze hledat vhodné mapování objektů. Mezi mapovanými úkoly bude docházet k synchronizaci časových záznamů obousměrně, jako je tomu nyní u spojení Redmine a Toggl Track.

Náročnost : Střední

Priorita : Must have

Dotčené části : Core, API, Frontend

Typ požadavku : Funkční – functionality, usability

SPN05 – Synchronizace časů i jiného uživatele

Nástroj Timer2Ticket bude mezi projektovými nástroji synchronizovat úkoly i všechny vykázané časy k úkolům, včetně úkolů jiných uživatelů, než je ten, který konfiguruje spojení. Uživatel konfigurující spojení musí tím pádem mít dostatečná práva v obou systémech, aby Timer2Ticket mohl následně tuto synchronizaci provádět. Dostatečná oprávnění budou kontrolována už při konfiguraci spojení.

Náročnost : Střední

Priorita : Must have

Dotčené části : Core, API, Frontend

Typ požadavku : Funkční – functionality

SPN06 – Mapování projektů a úkolů mezi projektovými nástroji

Nástroj Timer2Ticket bude párovat projekty a úkoly v nich pouze tehdy, pokud nebude pochyb o tom, že se skutečně jedná o stejné úkoly. Pro správné určení párů projektů bude uživateli zpřístupněno nastavení mapování ve webovém rozhraní aplikace.

Náročnost : Střední

Priorita : Must have

Dotčené části : Core, API, Frontend

Typ požadavku : Funkční – functionality, usability

SPN07 – Notifikace o chybách

Uživatel bude notifikován o chybách při synchronizaci vybraným notifikačním kanálem (Sentry, e-mail). Chybou je zde myšleno to, že se aplikaci Timer2Ticket nepodaří najít pár úkolů (v rámci projektu) k synchronizaci. V takovém případě synchronizace logicky nemůže proběhnout a uživatel se to musí dozvědět.

Náročnost : Střední

Priorita : Should have

Dotčené části : Core, API

Typ požadavku : Funkční – usability

SPN08 – Konfigurace synchronizace času bez úkolu

Problém s vykazováním času mimo úkoly vzniká kvůli nástroji Redmine, který umožňuje vykazovat čas pouze k projektu bez konkrétního úkolu. Ve webovém rozhraní aplikace bude uživatelům zpřístupněno nastavení pro konfiguraci, co dělat s časem vykázaným k projektu, nikoliv ke konkrétnímu úkolu. Možnosti výběru budou:

Takový čas zahodit – Časový záznam v takovém případě nebude synchronizován. Uživatel bude o této skutečnosti informován notifikací.

Vytvořit fallback úkol (v Jira) – ve druhém systému (konkrétně Jira) vznikne v každém projektu jeden nový úkol, do kterého budou synchronizovány všechny časy, které byly vykázané k projektu bez úkolu v Redmine. Tento úkol bude mít uživatel možnost v rámci spojení pojmenovat.

Základní a předvybranou volbou bude časové záznamy vůbec nesynchronizovat.

Náročnost : Střední

Priorita : Should have

Dotčené části : Core, API, Frontend

Typ požadavku : Funkční – usability

SPN09 – Konfigurace mapování uživatelů

Uživateli bude zpřístupněna možnost mapovat uživatele mezi systémy, aby byl synchronizovaný čas vykazován za správného člověka. V základním režimu nebude Timer2Ticket řešit uživatele a bude synchronizovat pouze čas. Pokud to bude uživatel vyžadovat, bude mít možnost ve webovém rozhraní Timer2Ticket párovat uživatele v jednom a ve druhém systému.

Náročnost : Střední

Priorita : Should have

Dotčené části : Core, API, Frontend

Typ požadavku : Funkční – usability

2.7 Vytížení zdrojů společnosti Jagu s. r. o. přepisováním časových záznamů

Společnost Jagu s. r. o. využívá interně projektový nástroj Redmine. Někteří jejich zákazníci ale používají jiné projektové nástroje, v této práci mě zajímá především nástroj Jira. Podle záznamů (příloha B) Sáry Sovičkové z několika posledních měsíců vyplývá, že vytvářením kopií úkolů z Jira do Redmine tráví asi dvě hodiny měsíčně. Přenášení časových záznamů mezi systémy se v tuto chvíli kvůli časové náročnosti nedělá.

Na zkoumaném projektu je údajně asi 50 nových úkolů každý měsíc. Pokud by měl být čas vykazován nejen v Redmine, ale i v Jira, tak by to znamenalo okolo 500 manuálních synchronizací měsíčně. Časových záznamů by totiž mělo být průměrně okolo 10 na úkol. Pokud každé přenesení času zabere zhruba minutu (otevření jednoho a druhého nástroje, vyhledání úkolů a přepsání časových záznamů), tak by to znamenalo zhruba 500 minut práce měsíčně, což odpovídá více než 8 hodinám. (výpočet z B)

Dohromady, spolu s dvěma hodinami potřebnými ke kopírování úkolů z Jira do Redmine, se dostáváme na $126 + 500 = 626$ minut, což je asi deset a půl hodiny, které může potenciálně ušetřit vhodná integrace mezi Jira a Redmine.

Kapitola 3
Návrh

V této kapitole rozebírám sesbírané požadavky a navrhuji, jak tyto funkcionality zakomponovat do Timer2Ticket. Navrhuji jak zakomponovat další projektový nástroj Jira, jak zkvalitnit uživatelský zážitek, jak přizpůsobit stávající vrstvy aplikace novým požadavkům, jak rozšířit datovou vrstvu a jak zakomponovat webhooks nebo přidat notifikace.
Výsledkem je popis, jak mají být zakomponovány výše zmíněně funkcionality a nový datový model a model architektury.

3.1 Přidání nástroje Jira

Přidání nástroje Jira do Timer2Ticket je podle požadavků z kapitoly 2 stěžejní částí této práce. Je proto třeba navrhnout fungování komunikace mezi Jira a Toggl Track i Jira a Redmine.

3.1.1 Komunikace

Timer2Ticket bude s Jira komunikovat přes REST API, protože je to jediný způsob, jak s Jira vzdáleně komunikovat. REST API má Jira navíc velmi bohaté a dobře zdokumentované. Zajímat mě budou především endpoints manipulující s úkoly (issues). [62]

3.1.2 Autentizace

Jira poskytuje několik možností Autentizace a Autorizace přes REST API, všechny popsané v dokumentaci [63].

První a jednoduchou možností na implementaci je využití „Basic Auth“. V takovém případě uživatel musí vytvořit API klíč, který bude poskytovat dostatečná oprávnění pro manipulaci s úkoly. API klíč, uživatelovu doménu [1] a e-mail by v takovém případě nahrál uživatel do webového rozhraní Timer2Ticket podobně, jako tomu je nyní u Toggl Track a Redmine. Aplikace Timer2Ticket by tímto dostala přístup k uživatelově Jira a mohla provádět synchronizaci.

Druhou možností je využití OAuth2.0 [64]. Uživatel by v takovém případě byl při konfiguraci spojení přesměrován z webu Timer2Ticket na přihlašovací stránku k Atlassian [2] účtu. Musel by se přihlásit a dát přístup nástroji Timer2Ticket. Následně by byl přesměrován zpět do konfigurace spojení na webu Timer2Ticket. Timer2Ticket pak bude mít přístup k manipulaci s potřebnými daty.

[1] Možno také nazvat jako API point
[2] Společnost Atlassian provozuje Jira

Vzhledem k časové náročnosti a nižšímu riziku implementace jsem vybral možnost autentizace přes API klíč. Bude ji snadnější implementovat a lépe zapadá do systému Timer2Ticket, protože je využívána i u stávajících aplikací a uživatelé tento způsob znají.

3.1.3 Klíče a id

Jira kromě přístupu přes id umožňuje u spousty objektů i přístup přes „klíč“ (atribut key). Ten je zpravidla lépe lidsky čitelný a také je unikátní v rámci jedné domény. Například pro můj projekt „Timer2Ticket“ je využit klíč „T2T“. Od tohoto klíče k projektu jsou pak následně odvozovány i klíče jednotlivých úkolů. Například mám úkol s klíčem „T2T-25“ a id „10026“.

Odhaduji, že klíč úkolu se odvozuje od klíče projektu složením klíče projektu a pořadovým číslem úkolu v projektu. Nikde jsem ale nedohledal pravidla pro odvozování a nemohu si tak tím být jistý. Nejen proto bude v implementaci vhodnější pracovat s atributem *id* spíš než s atributem *key*. Časové záznamy synchronizuje stroj a proto nevidím výhody v používání klíče. Naopak vím, že id je číslo a to mi usnadní případnou manipulaci s ním.

3.1.4 Práce s id cizí služby v TESO

TESO (Time Entry Synced Object) je objekt v databázi Timer2Ticket, který popisuje časový záznam. V rámci něj jsou uloženy pro každou službu objekty STEO (Service Time Entry Object), které se odkazují na svůj obraz v synchronizovaných službách. Schématicky je toto znázorněno na obrázku 2.2. Objekt STEO obsahuje pouze pole *id* a *isOrigin*, který říká, ve které službě tento záznam vznikl. Práce s časovým záznamem byla dosud možná přes unikátní id Časového záznamu (v Redmine i Toggl Track je id časového záznamu unikátní [65, 66]).

Jira toto ale neumožňuje, jelikož časový záznam je pouze součástí nějakého z úkolů a neexistuje jako samostatná entita. Id časového záznamu tak v Jira není unikátní (je unikátní v pouze rámci úkolu), a navíc k práci s časovými záznamy nevystavuje Jira vhodný API endpoint. Jednotlivé časové záznamy se nacházejí v atributu úkolu „worklog“, který obsahuje pole jednotlivých časových záznamů.

K přístupu k časovému záznamu v Jira je tedy potřeba nejen Id časového záznamu, ale i Id úkolu. Id úkolu ale není nyní uloženo v databázi Timer2Ticket a stávající implementace navíc nepočítá s tím, že by mělo být třeba.

Budu tedy nucen, při vytváření a ukládání časového záznamu do databáze, zakódovat do atributu *id* STEO nejen id časového záznamu, ale i id úkolu, ke kterému patří. Tento atribut je naštěstí typu *string*, takže je možné do něj zakódovat téměř cokoliv.

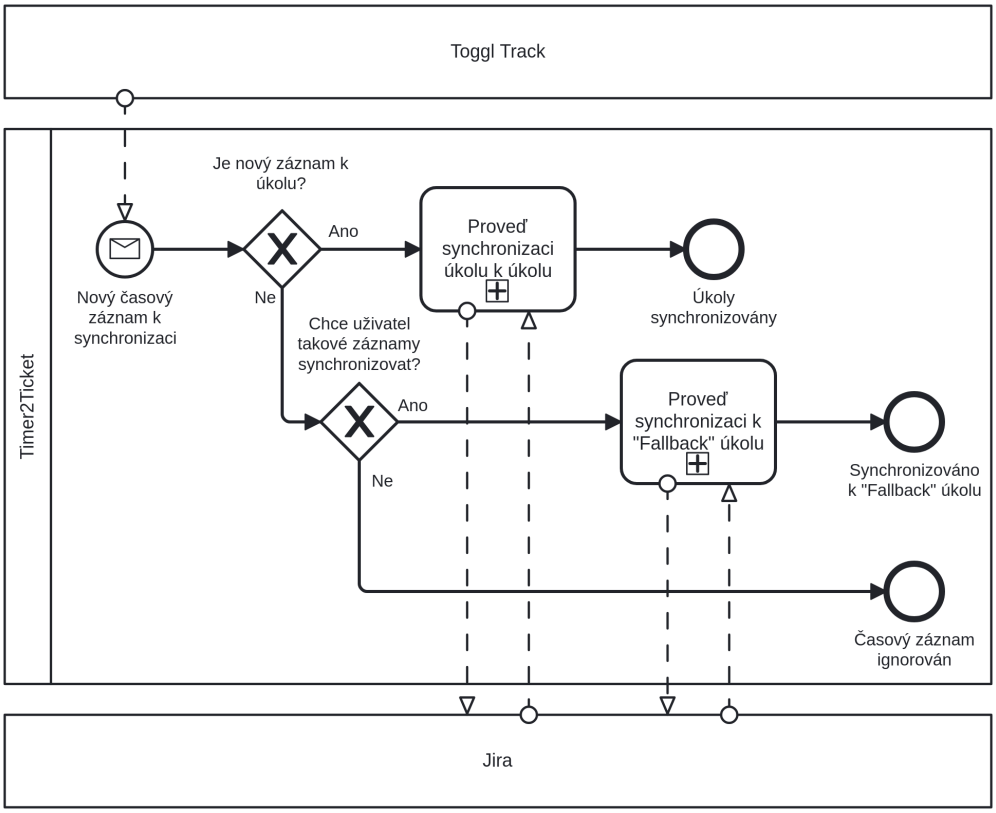
V Jira se jako id úkolů využívají výhradně čísla, mohu tedy zvolit libovolný nečíselný znak jako oddělovač. Budu tedy používat podtržítko pro oddělení 2 částí id STEO. Před prvním podtržítkem bude id úkolu z Jira a za dvojtečkou bude id časového záznamu v rámci úkolu. Celkově mi tak vznikne unikátní identifikátor časového záznamu ve formátu:

„idÚkolu“ _ „idČasovéhoZáznamu“

.

3.1.5 Přiřazování časových záznamů k úkolům

Aby bylo možné časové záznamy přidávat k úkolům v Jira, je třeba mít zapnutý „Time Tracking“ v nastavení projektu. Tato možnost zpřístupní v náhledu úkolu možnost „přidat čas“. Uživateli je sice v náhledu úkolu zpřístupněna pouze suma času stráveného na úkolu, ale v datech úkolu jsou jednotlivé záznamy, se kterými lze pracovat tak, jak bylo zamýšleno v původním návrhu Timer2Ticket [1]. Je také nutné, aby měl uživatel potřebná oprávnění k přidávání a mazání časových záznamů. Pokud bude některé z oprávnění chybět, skončí synchronizace chybou. Konkrétně je



Obrázek 3.1 Rozhodovací proces pro přiřazení časového záznamu z Toggl Track do Jira, vytvořeno pomocí [67], notace BPMN [68]

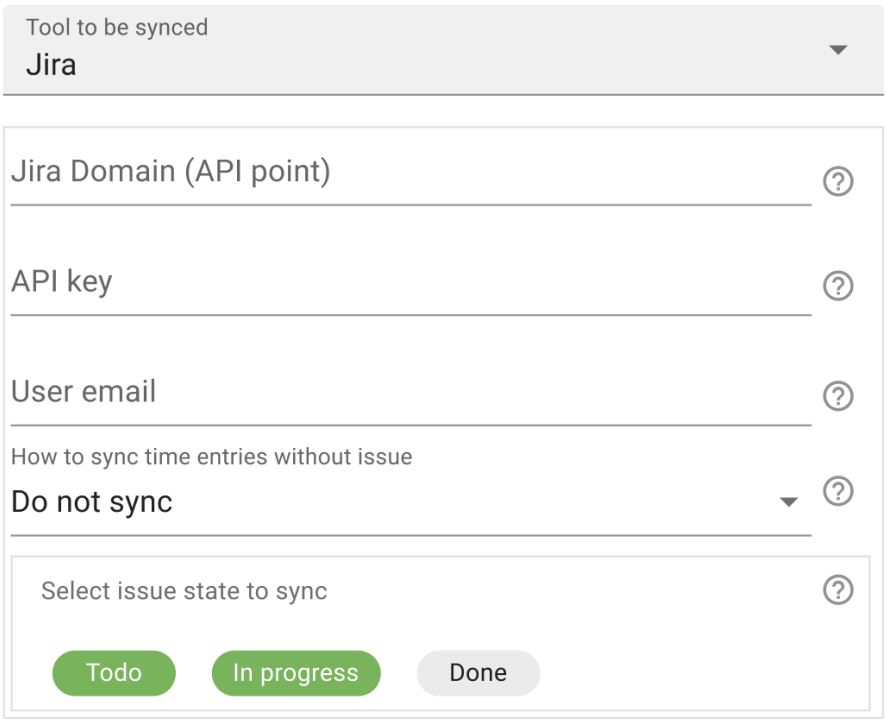
potřeba mít na paměti, že uživatel musí mít oprávnění mazat vlastní časové záznamy, které není v Jira samozřejmé.

Problém nastává se zaznamenáváním času k projektu. Redmine i Toggl Track toto umožňují, ale Jira nikoliv. Všechny časy v Jira zaznamenané musí být k některému z úkolů. Toggl Track ale umožňuje naměřit čas k projektu, proto je nutné vyřešit, kam s takovýmto časovým záznamem. Z požadavků vyplývá, že tento časový záznam nelze pouze ignorovat a je potřeba mít možnost ho k nějakému úkolu přiřadit. Uživatel bude mít možnost výběru mezi ignorováním synchronizace s notifikací, že takový časový záznam nebyl mezi nástroji synchronizován, nebo mu bude vytvořen úkol pro tyto časové záznamy (v každém projektu). Příklad, jak bude Timer2Ticket postupovat při novém časovém záznamu v Toggl Track je znázorněn na obrázku 3.1. Detailněji rozebírám možnosti výběru pro časové záznamy bez úkolu v sekci návrhu konkrétního formuláře v části 3.1.7.

3.1.6 Uložení id uživatele

Timer2Ticket nyní v rámci spojení (Connection) ukládá pro každou synchronizovanou službu do databáze *userId*, které reprezentuje id uživatele v synchronizované službě. Atribut je typu *number*, ale Id uživatele v Jira je *string*. Bude proto nutné upravit databázové schéma tak, aby bylo možné ukládat userId ze synchronizované služby jako *string*.

Druhou možností je udělat tento atribut povinný pouze pro služby Redmine a Toggl Track, protože všechny operace s Jira je možné vyřídit bez uživatelova id. Místo něj je uživatel unikátně identifikován e-mailem. Ten je i tak potřeba k autentizaci a je v rámci domény stejně unikátní



Obrázek 3.2 Návrh konfigurace spojení Jira, vytvořeno pomocí [69]

jako id.

Atribut *userId* sice v tuto chvíli pravděpodobně nebude pro fungování spojení s Jira potřeba, lze ho ale přes Jira API získat při klasické kontrole zadaných uživatelských údajů, takže není pro implementaci žádnou přítěží si toto id pamatovat. A proto bude ukládáno do Timer2Ticket databáze s tím, že proběhne změna typu atributu v databázi.

3.1.7 Přidání konfigurace Jira do webového rozhraní

Ve webovém rozhraní Timer2Ticket musí vzniknout formulář pro konfiguraci Jira. Od uživatele je třeba získat API Klíč, Jira doménu (API point) a přihlašovací e-mail. Navržený vzhled bude vycházet ze stávajícího návrhu z bakalářské práce Jakuba Čermáka [4].

Základní konfigurace spojení

Základní konfigurace spojení bude od uživatele vyžadovat vyplnění textových polí pro Jira doménu, API klíč a jeho přihlašovací e-mail. Struktura bude velmi podobná struktuře konfigurace u stávajících služeb. Všechna tato pole budou povinná. Navíc ale bude nutné vybrat, jak se má přistupovat k času, který byl v Toggl Track naměřen k projektu (viz 3.1.7).

U stávajících služeb proběhne po zadání povinných údajů kontrola spojení a jsou stahovány další konfigurační možnosti (workspace pro Toggl Track a výchozí aktivita pro Redmine). Stav kontroly je uživateli zobrazován u jednotlivých polí (načítající kolečko, zelená fajfka, červený křížek). Jira se musí v tomto chovat stejně. Po zadání domény, API klíče a e-mailu proto proběhne kontrola spojení s Jira. V případě úspěchu se uživateli zpřístupní další možnosti nastavení, v případě neúspěchu bude uživatel upozorněn na chybu při navazování spojení a bude vyzván ke kontrole údajů. Celkový návrh formuláře pro konfiguraci Jira je na obrázku 3.2.

Nastavení časových záznamů k projektu

Po úspěšném navázání spojení mohou být stažena data pro další nastavení. Na rozdíl od Redmine neumožňuje Jira vykazovat naměřený čas k projektu bez přiřazeného úkolu. V Toggl Track je ale možnost takový čas naměřit a uživatel proto dostane při konfiguraci na výběr, co s takovým časovým záznamem má Timer2Ticket dělat.

Časový záznam bez tagu nesynchronizovat – Výchozí varianta, při které nebude takový záznam z Toggl Track do Jira vůbec přenášen. Takový záznam pravděpodobně vznikl chybou uživatele (zapomněl vybrat úkol, ale vybral projekt). Proto bude o takovém časovém záznamu notifikován, aby ho měl možnost opravit.

Timer2Ticket založí v každém projektu úkol pro takové záznamy – V případě potřeby synchronizovat takový čas, bude v Jira založen nový úkol, do kterého následně budou takové záznamy synchronizovány. Uživateli bude umožněno takový úkol pojmenovat podle sebe.

Další možností by mohlo být, dát uživateli vybrat úkol, do kterého má být takový záznam zařazen. Uživatel by ale musel takový úkol vybrat pro každý projekt. Obzvláště pokud by existovalo v Jira mnoho projektů, tak by se konfigurace výrazně protáhla a uživatelsky znepříjemnila. Navíc by nastal problém v případě založení nového projektu. Uživatel by s každým novým projektem musel jít upravit konfiguraci do Timer2Ticket, což není žádoucí. Z těchto důvodů jsem prozatím zavrhl nastavování pro každý projekt zvlášť.

Nastavení synchronizace úkolů podle stavu

Uživateli bude umožněno vybrat stavy úkolů, které mají být synchronizovány. Například tak uživatel bude mít možnost říct, že nechce synchronizovat hotové úkoly. Pokud úkol změní stav ze stavu, který měl být synchronizován do stavu, že synchronizován být nemá, bude odstraněn ze druhé služby, a nebude tak možné vybrat odpovídající úkol v nástroji pro měření času. Synchronizace časových záznamů se tento výběr vůbec nedotkne, zde se jedná o synchronizaci úkolů na jednotlivé tagy.

Ve výchozím stavu budou všechny stavy nastaveny tak, že mají být synchronizovány a uživatel bude mít možnost synchronizaci u stavů zrušit. Výběr bude uživatel provádět označováním čipů (chip) jednotlivých stavů. Pokud vznikne v projektovém nástroji nový stav pro úkoly, bude automaticky synchronizován. V nastavení spojení pak bude možné synchronizaci pro tento stav vypnout.

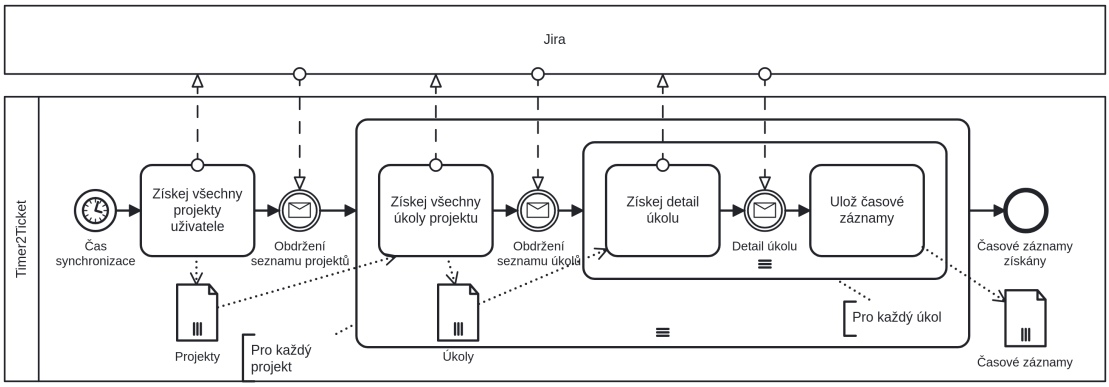
Jinou možností by bylo umožnit uživateli zadat vlastní JQL [70] výraz. Takto by měl uživatel větší svobodu v synchronizaci, ale zároveň by to znamenalo násobně složitější validaci na straně Timer2Ticket. Navíc není jasné, kolik uživatelů by takovou možnost využívalo. Proto zatím kvůli jednoduchosti uživatelského rozhraní bude tato možnost vynechána.

Návrh webového rozhraní pro výběr stavů je znázorněn na obrázku 3.2.

3.1.8 Získání projektů a úkolů přes API

Jira bude v tomto spojení v roli projektového nástroje („Ticket“), bude proto považována za zdroj pravdy o úkolech. Úkoly a projekty z Jira budou přenášeny do nástroje pro měření času („Timer“), jak je znázorněno na obrázku 2.10. Jednotlivé časové záznamy pak budou synchronizovány obousměrně.

Pro získání seznamu projektů bude využit příslušný endpoint */rest/api/3/project*. Získávání úkolů (issues) bude ale složitější. Jira API totiž nemá vystavený endpoint pro získání všech issues, nebo všech issues projektu. Issues je možné získat s příslušným unikátním issue *id* nebo *Key*. Kde vzhledem k doméně jsou obě unikátní a odvozené od projektu, ve kterém se nachází (například klíč vzniká pravděpodobně jako „jménoProjektu“-„pořadíIssue“). Nedohledal jsem ale nikde tyto



Obrázek 3.3 Získání všech časových záznamů přes Jira API, vytvořeno pomocí [67], notace BPMN [68]

pravidla a nerad bych vsázel na to, že budu schopen správně odvodit identifikátor všech issues. Navíc není jasné, kolik úkolů v danou chvíli je.

Jako lepší varianta se mi jeví pomocí endpointu pro vyhledávání $/rest/api/3/search?jql = project = ABC$ s využitím query v jazyce JQL [70]. Tento příkaz vrací všechny issues z projektu ABC. Toto řešení bylo doporučeno uživateli se stejným problémem na diskusním fóru Atlassian [71]. Stačí tedy tento dotaz upravit, aby vrátil issues všech projektů (které už znám z dotazu na projekty), nebo zavolat dotaz vícekrát. Jazyk JQL navíc umožňuje dobré možnosti vyhledávání, lze ho tak například využít k vyhledání issues jen určitou dobu zpátky, nebo s příslušným stavem (a nesynchronizovat hotové).

3.1.9 Získání časových záznamů přes API

Časové záznamy v Jira neexistují jako samostatná entita, ale jsou součástí jednotlivých úkolů jako atribut *worklog*. Job synchronizace časových záznamů (kapitola 2.6 [1]) potřebuje k provedení synchronizace všechny časové záznamy (omezeno stářím na 60 dnů) z obou služeb. Pak v nich vyhledá ty, které nejsou spárované s některým ve druhé službě a vytvoří tak páry. Pro získání všech časových záznamů přes Jira API je nutné zeptat se postupně na všechny úkoly a následně prozkoumat jejich časové záznamy. K tomu využiji endpoint $/rest/api/3/issue/\{issueIdOrKey\}$. Znamená to tedy, že pro získání časových záznamů projektu s 50 úkoly potřebuji více než 50 API volání. Schématicky je tento postup znázorněn na obrázku 3.3.

Protože úkolů může být v projektu mnoho, a ne všechny obsahují nové časové záznamy, využiji při jejich hledání (viz 3.1.8) navíc query parametr *worklogDate*, který umožňuje filtrovat podle data změny ve worklogu. Momentálně jsou synchronizovány nové záznamy 60 dní zpět, takže nastavením této hranice omezím úkoly na ty, kterým se měnil worklog v posledních 60 dnech. Doufám, že tím omezím množství nutných API volání.

3.2 Spojení dvou projektových nástrojů

Přidáním nového projektového nástroje do Timer2Ticket (Jira) vzniká potřeba umožnit synchronizaci časových záznamů i mezi projektovými nástroji. Z požadavků SPN01-SPN09 2.6.4 vyplývá, že nebudou mezi nástroji navzájem vytvářeny úkoly, ale Timer2Ticket musí nalézt vhodné mapování objektů. Synchronizace časových záznamů pak bude probíhat podle standardního schématu.

Cílem synchronizace mezi projektovými nástroji je usnadnit práci vývojového týmu v situaci, kdy aktuální zákazník využívá jiný projektový nástroj, než vývojový tým. Úkolem Ti-

mer2Ticket potom bude především synchronizace časových záznamů směrem z projektového nástroje vývojového týmu do projektového nástroje zákazníka. Musí ale být schopen reagovat i na případnou změnu v časových záznamech v zákazníkově systému a správně je propsat do druhého systému.

Nástroj Timer2Ticket bude umět synchronizovat časové záznamy i mezi stejnými projektovými nástroji (Jira-Jira, Redmine-Redmine). Stejné nástroje ale bude možné synchronizovat pouze v případě, že jsou provozovány na rozdílných doménách. Synchronizaci dvou různých projektů v rámci jedné domény Timer2Ticket řešit nebude.

3.2.1 Mapování úkolů mezi systémy

Mapovat jednotlivé úkoly na sebe ručně ve webovém rozhraní aplikace Timer2Ticket nedává smysl, protože úkolů může být velké množství a budou vznikat často. To by nutilo projektového manažera pravidelně chodit do Timer2Ticket, což popírá prvotní záměr nástroje. Je tedy třeba nalézt způsob, jak má Timer2Ticket v rámci projektu nalézt správné dvojice úkolů.

Vzhledem k požadavku SPN01B, který zakazuje vytváření úkolů v projektových nástrojích, budu počítat s tím, že úkoly existují v obou systémech 1:1 a úkolem Timer2Ticket je pouze tyto dvojice najít bez vytváření dalších úkolů. Pokud se některému úkolu dvojici najít nepodaří, bude o tom uživatel notifikován.

Mapování podle společného atributu

Úkoly v obou systémech budou nepochybně mít své id a název. Id budou nepochybně odlišné, ale pokud by se úkoly jmenovaly v obou systémech stejně, lze to pro mapování využít. Shoda názvu ale není zaručena. Součástí přenesení úkolu z jednoho do druhého nástroje může být i změna názvu tak, aby lépe odpovídal zvyklostem týmu. Proto nevidím mapování podle jména úkolu jako vhodné.

Mapování podle vlastního pole (custom field)

Projektové nástroje umožňují definici vlastního pole. Toto pole lze využít například pro vložení id požadavku ve druhém nástroji a Timer2Ticket by následně měl práci triviální. Přečetl by id z definovaného custom field a vytvořil tak dvojici úkolů. Nevýhodou je, že toto klade vyšší nároky na uživatele. Ti by museli při zakládání úkolu v interním nástroji zkopírovat navíc id úkolu z nástroje zákazníka.

Aktuální flow zadavatele je takové, že při zakládání nového úkolu v interním Redmine je přidán odkaz na tento úkol do zákazníkovy Jira. Tím vzniká jednoznačné provázání úkolů (i když jenom směrem z Jira do Redmine). Odkaz na úkol v sobě vždy (alespoň u Redmine a Jira) obsahuje i identifikátor daného úkolu. Timer2Ticket tak může z úkolu v jednom z nástrojů získat informace o jeho dvojníkovi v nástroji druhém.

Odkaz na úkol v Jira je vždy ve formátu .../$\{issueKey\}$ nebo .../$selectedIssue = \{issueKey\}$ a v Redmine .../$\{issueId\}$. Pro oba nástroje by tedy fungovalo oříznout URL na posledním lomítku. Následně by v Timer2Ticket vzniklo provázání úkolů a to umožní následnou synchronizaci časových záznamů.

Pokud se náhodou podaří uživateli získat odkaz na úkol jiným způsobem a URL nebude mít na konci id, označí to Timer2Ticket za chybu a nebude s tímto úkolem pracovat. Uživatel bude samozřejmě notifikován.

Průběh synchronizace

Ve chvíli kdy bude Timer2Ticket mít vytvořené mapování úkolů, tak bude synchronizace probíhat stejně jako nyní. Ve vrstvě Core bude v pravidelných intervalech spouštěn job pro synchronizaci

časových záznamů. Ten bude nadále vytvářet nové časové záznamy, aktualizovat změněné a případně mazat odstraněné. Jediný rozdíl může nastat ve způsobu vykazování, protože ve spojení bude figurovat více uživatelů (viz sekce 3.2.3).

Změna mapovacího pole

Pokud uživatel změní obsah mapovacího pole v době, kdy Timer2Ticket už toto mapování úkolů vytvořil, tak se změna nepropíše do synchronizace. Umět reagovat na takovou změnu by totiž znamenalo zásadně zasáhnout do synchronizačního jobu, který by musel před synchronizací kontrolovat aktuálnost všech mapování. To by znamenalo získat každý úkol přes API a kontrolovat mapovací custom field. V případě změny by musely být odstraněny všechny synchronizované časové záznamy těchto úkolů, nalezeno nové mapování a vše by muselo být synchronizováno podle něj. Kvůli tomu jsem se po konzultaci s Oldřichem Malcem rozhodl, že Timer2Ticket na případnou změnu mapování úkolu nebude reagovat.

3.2.2 Mapování projektů

Výběr projektů k synchronizaci a jejich mapování bude provádět uživatel ve webovém rozhraní Timer2Ticket při konfiguraci synchronizace. Konfiguraci bude provádět za celý tým pouze jeden jeho člen (projektový manažer). Kromě standardního vyplnění přístupů pro spojení (API klíč, API point, ...) bude muset provést výběr projektů k synchronizaci a definovat jejich párování.

Timer2Ticket není totiž sám schopný rozumně říct, jak mají být projekty mapovány. Mohl by například hledat shodu ve jméně projektu, ale ta nemusí být stoprocentní a navíc nebude schopen říct, zda mají být tyto projekty součástí synchronizace. Proto bude tato práce na uživateli.

Přidání mapování projektů do webového rozhraní

Při konfiguraci spojení mezi dvěma projektovými nástroji bude nutné definovat i mapování projektů a vybrat custom field, přes který bude docházet k mapování úkolů. Identifikoval jsem tři možnosti, v jakém pořadí projekty a custom field vybírat.

Custom field a následně páry projektů – Uživatel by nejdřív vybral mapovací custom field a následně by dostal k výběru pouze takové projekty, které tento custom field mají. Z nich by následně tvořil páry.

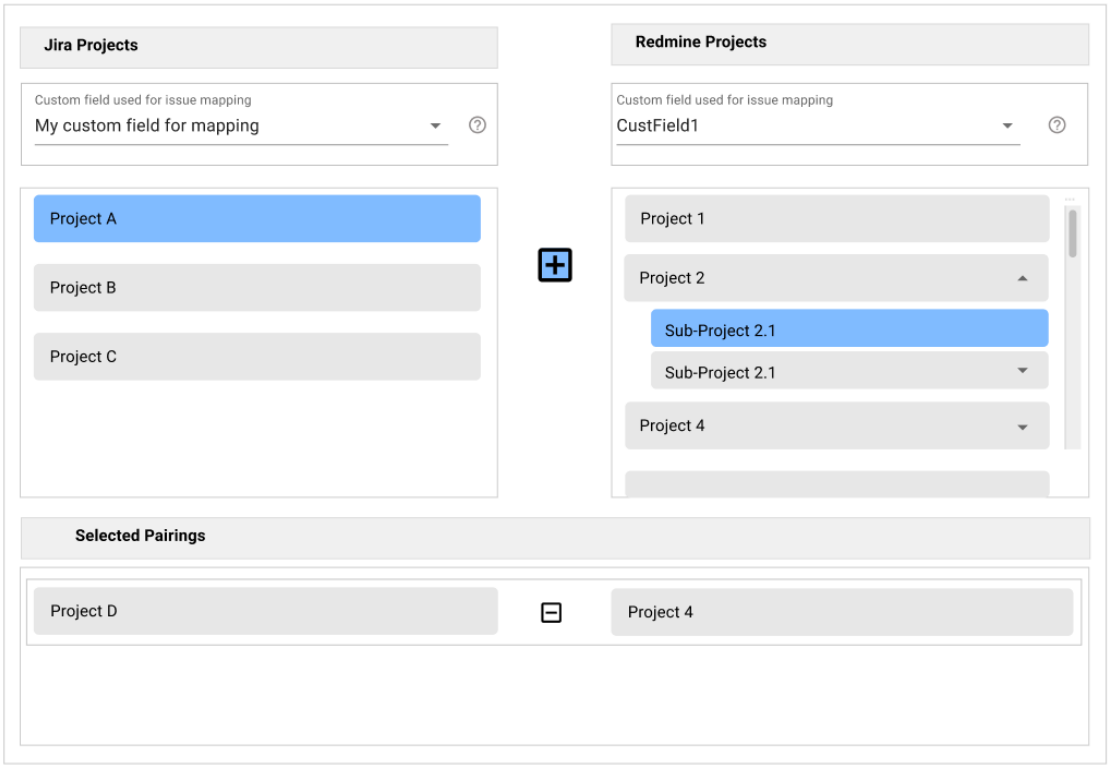
Projekty a následně custom field – Nejdřív by uživatel vytvořil páry projektů a následně by vybral mapovací custom field z průniku vybraných.

Jiný custom field pro každý pár projektů – Pro každý pár projektů bude nutné vybrat mapovací custom field. Každý pár tak bude moct využívat jiný mapovací atribut.

Rozhodl jsem se pro variantu nejdříve zvolit mapovací custom field a následně nabídnout uživateli projekty, které tento custom field mají. Pro uživatele to bude přehlednější, protože pokud nenajde projekt, který hledá, tak příčinou bude chybějící custom field. Může ho tak v projektovém nástroji přidat a nechat si aktualizovat projekty v konfiguraci spojení. Pokud by vybíral mapovací custom field až po výběru projektů, tak by mohl být zmatený, že hledaný custom field není v nabídce, protože ho jeden z projektů nemá. Uživatel by pak ale neměl jak zjistit, který z projektů je ten problematický.

Volit mapovací custom field pro každý pár projektů by bylo rozhodně nejflexibilnější, ale zároveň nejpracnější pro uživatele. Pokud bude mít uživatel potřebu synchronizovat projekty podle jiného custom field, bude muset využít jiné spojení.

Mapování projektů a custom field bude nutné definovat už při prvotní konfiguraci spojení a nepůjde bez něj spojení vůbec uložit. Synchronizaci jednotlivých projektů bude následně možné přidávat i odebírat, ale už nebude možné měnit mapovací parametry (custom field) kvůli



Obrázek 3.4 Návrh mapování projektů, vytvořeno pomocí [69]

problémům popsaných v 3.2.1. Kompletní návrh mapování projektů je k dispozici na obrázku 3.4.

Mapování projektů a pod-projektů

Redmine umožňuje v rámci projektů vytvářet pod-projekty. Při konfiguraci spojení musí být uživateli umožněno mapování nejen projektů, ale i pod-projektů. Mapování bude 1:1 s tím, že z každého synchronizovaného nástroje může být vybrán jako projekt k párování i libovolný pod-projekt. Je tedy třeba uživateli nabídnout možnost prostupovat v Redmine stromu projektu k jeho pod-projektům a umožnit mapování takového pod-projektu na vybraný projekt z druhého nástroje. Návrh takového párování je k dispozici na obrázku 3.4.

3.2.3 Mapování uživatelů

Aby bylo možné vykázaný čas přenést správně za jednotlivé uživatele, musí existovat mezi projektovými systémy jejich správné mapování. Problém je, že ne všichni uživatelé musí mít účet v obou projektových systémech. Nastavení mapování uživatelů bude probíhat již při konfiguraci spojení. Nastavení bude možné měnit i u běžící synchronizace, ale pak nedojde ke zpětnému přepsání.

Nemapovat uživatele

Nejjednodušší možnost mapování uživatelů je taková, že žádní uživatelé mapováni nebudou. Jakýkoliv vykázaný čas pak bude synchronizován tak, jako by ho vykázal uživatel konfigurující

spojení. Toto řešení je triviální na konfiguraci a je funkční co se týče celkové sumy synchronizovaného času. Proto bude zvoleno jako výchozí. Pokud ale nebude uživatel spokojený s tímto způsobem mapování, bude mu umožněno zvolit mapování uživatelů manuálně.

Mapování uživatelů 1:1

Pro potřeby mapování uživatelů bude ve webovém rozhraní Timer2Ticket zpřístupněno nastavení mapování uživatelů pro spojení. Cílem toho bude vytvořit páry uživatelských účtů (Jira účet – Redmine účet) pro jednotlivé členy projektového týmu. Budu cílit na co největší podobnost s mapováním projektů. Na rozdíl od projektů je zde ale nemalá šance na automatické spárování, protože například uživatelův e-mail bude často stejný v obou nástrojích. Timer2Ticket bude tedy schopen doporučit některé páry účtů sám a na konfigurujícím uživateli už nechat jen finální ladění.

Je třeba ale pamatovat na to, že ne všichni uživatelé musí mít v obou systémech účet. Proto zde musí být možnost vybrat výchozího uživatele pro ty, kteří nebudou mít mapování.

Konfigurace mapování uživatelů bude vzhledově podobná párování projektů (obrázek 3.4). Uživatelské páry identifikované nástrojem Timer2Ticket podle shodného e-mailu budou seřazeni na začátku výběru a odpovídající e-maily budou na stejném řádku. Projektový manažer tak bude mít k dispozici návrh, ale bude muset explicitně vytvořit dvojici. Předejdu tak potenciálním chybám oproti variantě, že bych přímo takové účty spároval a dal možnost je z výběru odstranit.

Umístění konfigurace mapování uživatelů

Mapování uživatelů již není klíčové pro správnou synchronizaci a je možné ho nastavit bez větších problémů později. Proto vznikne konfigurační box podobný konfiguraci rozvrhu synchronizace. Bude přístupný pouze u synchronizace služeb, kde dává smysl uživatele mapovat (Jira–Redmine ano, Jira–Toggl ne). Pokud uživatel mapování uživatelů nenastaví, bude všechen čas synchronizován jako by ho vykázal uživatel konfigurující spojení v Timer2Ticket.

Vykazování za jiného uživatele v Jira

V Jira bohužel není možné přes API vykazovat za jiného uživatele. Na takovou funkcionalitu už dlouho existuje požadavek [72]. Ten ale zatím není implementován.

Problém s vykazováním za jiného uživatele lze obejít pomocí dostupných pluginů (doplňků), například *Timesheets* [73]. Není ale rozumné, a ani vhodné, podmiňovat fungování nástroje Timer2Ticket používáním Jira doplňku. Proto využití *Timesheets* ani jiného podobného rozšíření nepřipadá v úvahu.

K vykázání času je proto potřeba mít přihlašovací údaje (API klíč a e-mail) daného uživatele. Timer2Ticket by tak musel získat tyto údaje i od dalších uživatelů, jejichž čas by měl být synchronizován. To by bylo možné implementovat přidáním uživatelů ke spojení. Timer2Ticket by následně mohl vybrat správného uživatele, přes kterého by synchronizace probíhala.

Při párování uživatelů bude projektový manažer (nebo jiný uživatel konfigurující spojení) vyzván k tomu, aby pro každého uživatele v Jira přidal jeho API klíč (e-mail lze získat přes API podle lidí na projektu). V případě, že bude třeba pro uživatele doplnit API klíč, vytvoří Timer2Ticket popUp okno pro zadání API klíče. PopUp volím proto, aby měl Timer2Ticket čas zkontrolovat správnost API klíče a nepustil uživatele v konfiguraci dál se špatně vybraným uživatelem. Návrh popUpu je k náhledu na obrázku 3.5.

Timer2Ticket si následně API klíč uloží ke spojení a bude ho používat při synchronizaci časových záznamů jednotlivých uživatelů. S tímto problémem by nepomohla ani implementace ověření uživatele přes OAuth [64], protože i tímto způsobem se jedná o přihlášení jednoho uživatele.

Obrázek 3.5 PopUp k zadání API klíče dalšího uživatele, vytvořeno pomocí [69]

Vykazování za jiného uživatele v Redmine

V Redmine je možné vykazovat čas za jiného uživatele, je k tomu ale třeba mít práva „Správa Času“ na projektu. Pokud by uživatel konfigurující spojení tato práva měl, tak bude možné snadno přenášet čas i za kolegy.

3.2.4 Spojení dvou projektových nástrojů nastavené více uživateli

Může se stát, že spojení mezi stejnými projektovými nástroji nakonfiguruje více uživatelů. Timer2Ticket by při konfiguraci mohl kontrolovat, zda už nemá v databázi spojení mezi vybranými doménami a případně další „stejné“ spojení nepovolit. To není rozumné, protože pak by mohl některý z uživatelů zablokovat některé spojení a další by se k tomu už nedostali. Navíc informovat uživatele chybovou hláškou „Dané spojení nelze nakonfigurovat, protože ho už používá někdo jiný.“ není ve vztahu k uživateli vhodné.

Dvě spojení mezi doménami mohou existovat vedle sebe bez jakýchkoli kolizí, pokud synchronizují jiné projekty. Pokud by ale měly synchronizovat stejné projekty, může dojít k několika problémům. První z nich je, že při synchronizaci časových záznamů budou nad těmito dvěma identickými spojeními spuštěny dva rozdílné joby s vlastním mapováním úkolů. Toto mapování je sice shodné, protože bylo nalezeno stejným algoritmem, ale každý job bude synchronizovat nad jiným spojením. Ohledně mapování tak vzniknou v databázi duplicity (každé spojení bude mít vlastní TESO pro každý časový záznam).

Oba joby potřebují vlastní mapovací objekty TESO a STEO. První job tedy vytvoří a smaže příslušné časové záznamy, vytvoří si příslušné mapovací objekty a skončí. V pořadí druhý job (joby se spouští sériově) zjistí, že všechny časové záznamy z jedné službě existují už ve službě druhé a opačně. Vytvoří si proto pouze mapovací objekty a skončí. V databázi tak pro různá spojení vznikají identické mapovací objekty.

Druhým a závažnějším problémem je to, který uživatel bude autorem vykázaného času. Bude to ten, jehož job bude spuštěn pro daný časový záznam první. Suma časů by měla být po synchronizaci stejná, ale autor synchronizovaného časového záznamu bude vždy z pohledu jednoho ze spojení špatný. Pokud navíc připustíme mapování uživatelů, kdy mohou být mapováni v každém spojení jinak (ne třeba záměrně, ale například rozdílným fallback uživatelem), tak bude pro uživatele velmi těžké pochopit, proč je daný časový záznam veden právě pod tímto uživatelem. Zase bude záležet na tom, který z jobů proběhne první.

Částečným řešením může být informování uživatele o již existujícím spojení, ať si podruhé rozmyslí, zda ho chce zprovoznit a upozornit ho, že by mohlo být rozumné si s kolegu zkontrolovat, kdo který projekt synchronizuje. Timer2Ticket sám takovému spojení ale rozumně zabránit

nedokáže a v případě duplicitního spojení nedokáže zařídit správného uživatele pro vykázání času.

3.3 Podpora webhooks

Pomocí webhooks by mohla synchronizace časových záznamů mezi nástroji probíhat prakticky okamžitě, navíc mohou Timer2Ticket pomoci v efektivitě kvůli omezení počtu nutných API dotazů na jednotlivé synchronizační služby. Obzvláště třeba v Jira, kde je pro získání časových záznamů potřeba obvolat po jednom všechny úkoly v každém projektu, by došlo k výraznému ušetření výpočetního výkonu.

Pro Timer2Ticket by to znamenalo vystavit endpoint pro zachytávání webhooks. Z pohledu uživatele je nutné nastavit webhook v synchronizované službě, kde zadá pouze URL Timer2Ticket endpointu. Timer2Ticket pak zvládne konfiguraci webhook přes API služby, kde bude třeba nastavit, které události má synchronizovaná služba hlásit.

3.3.1 Webhooks v Jira

Webhook uživatel nastaví v rozhraní aplikace, kde definuje URL, kam mají být v případě nějaké události zaslány data. Nastavení webhook pak probíhá přes API, kdy si Timer2Ticket zvládne webhook konfigurovat sám. Což je velká výhoda, protože je vyžadován pouze malý zásah od uživatele a má tak prostor udělat méně chyb.

Přidání webhook probíhá v nastavení Jira/Systém/Webhooks a je vyžadováno pouze URL. Následně je možné odchytávat změny v projektech, úkolech i v časových výkazech. U těchto objektů bude určitě třeba poslouchat na vytvoření, změnu i smazání. Pro všechny objekty jsou tyto události v Jira webhooks implementovány.

3.3.2 Webhooks v Toggl Track

Přidání webhook do Toggl Track je možné v uživatelském rozhraní (integrations/webhooks), nebo přes API. K nastavení přes API je dostačující API klíč, který Timer2Ticket už od uživatele má [60]. Nastavovat uživateli webhook bez jeho vědomí by ale nebylo správné, proto se Timer2Ticket uživatele při konfiguraci spojení na tuto možnost zeptá a v případě souhlasu nastaví webhook za něj.

Přes API je možné nastavit webhooks přesně podle potřeb služby, lze se nechat notifikovat o změnách v projektech, úkolech i časových záznamech. Pro potřeby Timer2Ticket by zatím byly časové záznamy dostačující, protože Toggl Track nebude možné využívat jako projektový nástroj.

3.3.3 Webhooks v Redmine

V Redmine webhooks nejsou podporovány. Toto téma řešil už ve své práci Martin Paul [5] a od té doby nedošlo k žádnému posunu (viz Redmine issue [74]).

3.3.4 Úprava členství kvůli webhooks

Webhooks nabízí okamžitou synchronizaci. Pokud by to bylo umožněno všem uživatelům, tak ztratí smysl aktuální monetizační model podle práce Jakuba Čermáka [4]. Proto přicházím s návrhy, jak toto řešit.

Modifikace členství Senior

Uživatelům s nejlepším členstvím Senior bude umožněno zprovoznit webhooks a tím získají okamžitou synchronizaci u služeb, kde je to možné. Tito uživatelé již mají tu nejlepší verzi Timer2Ticket, proto není nutné toto členství nijak omezovat.

Modifikace nižších členství Hobby a Junior

Uživatelům s levnějším členství Hobby a Junior bude umožněno webhook nakonfigurovat stejně jako pro uživatele s členstvím Senior. Timer2Ticket nebude ale v takovém případě zpracovávat webhooks okamžitě, ale bude je pro pozdější zpracování odkládat do fronty (viz 3.3.5).

Tito uživatelé ale nemají motivaci webhooks nastavovat. Pro provozovatele by to ale bylo výhodné, kdyby alespoň u Jira měli i tito uživatelé webhooks aktivované. Získávání časových záznamů z Jira je totiž časově náročné (velké množství API volání viz 3.1.9). Motivací jim proto bude dvakrát častější synchronizace, než je aktuálně v rámci jejich členství. Pro Hobby uživatele to bude dvakrát týdně a pro uživatele s členstvím Junior dvakrát denně, pokud webhooks zprovozní.

3.3.5 Fronta nezpracovaných webhooks

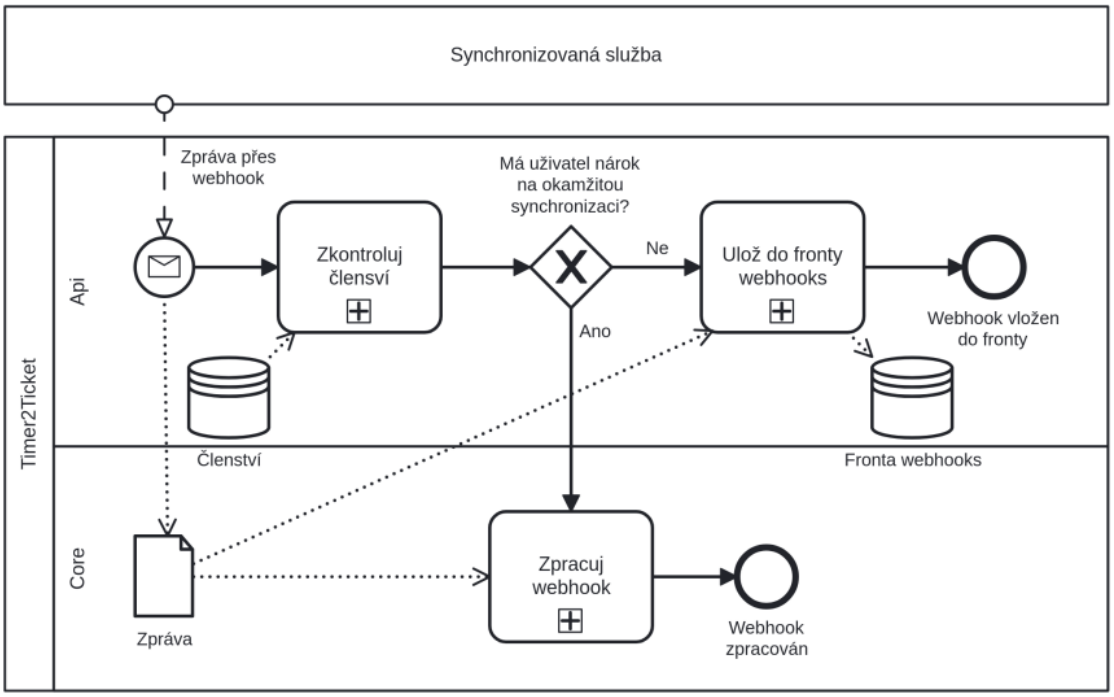
Kvůli tomu, že ne všichni uživatelé mají zaplacenou okamžitou synchronizaci, bude třeba někam odkládat obdržené zprávy, které ale nemají být uživateli zatím zpřístupněny. Za tímto účelem vznikne v databázi kolekce *WebhookQueue*, kam bude uložen objekt z příchozího webhook, který má být zpracován později. Při pravidelné synchronizaci tak Timer2Ticket nezačne okamžitě kontaktovat synchronizované služby, ale nejdřív zjistí, zda je na tomto spojení aktivní webhook (alespoň na jedné ze služeb). Pokud ano, tak Timer2Ticket získá potřebná data k synchronizaci z interní databáze místo přes API služby. Postup pro zpracování webhook je znázorněn na obrázku 3.6.

Kvůli tomuto přístupu bude nutné upravit i synchronizační joby. Nyní se totiž chovají tak, že porovnávají objekty mezi službami. Nově bude nutné mít možnost aplikovat změny (z WebhookQueue) na aktuální stav. Webhooks budou vždy pouze doplňkem k synchronizaci služeb přes API volání a jejich implementace nesmí ovlivnit stávající synchronizace. Poskytují ale možnost pro rychlejší a efektivnější synchronizaci. Nové schéma synchronizace je znázorněno na obrázku 3.7.

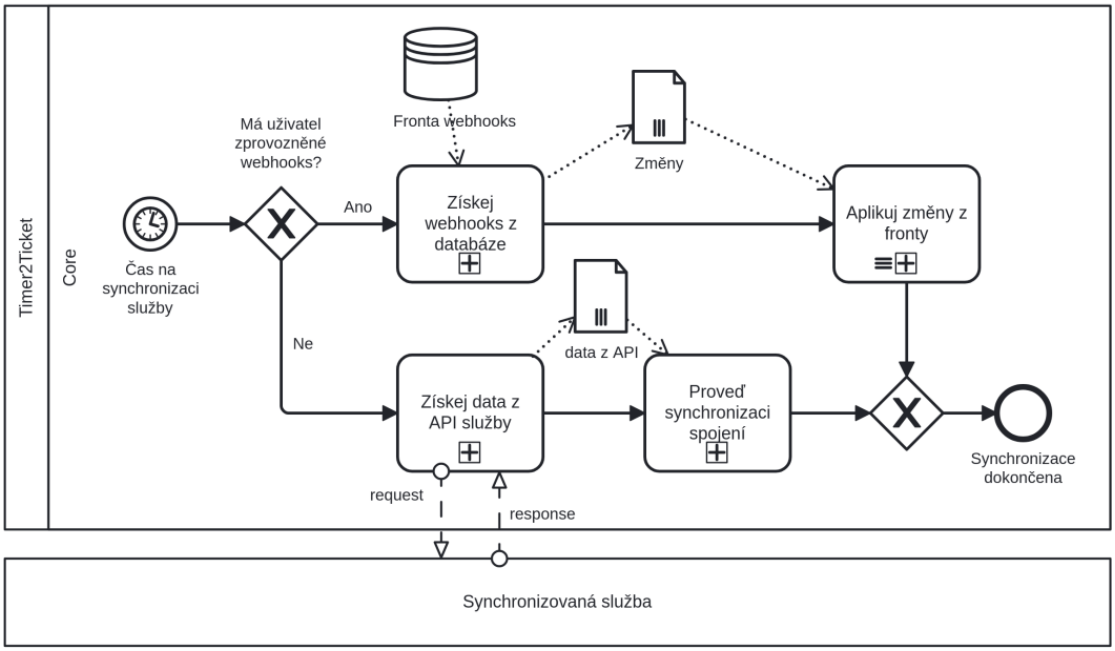
3.3.6 Vystavení endpoint pro webhooks

V rámci vrstvy Timer2Ticket Api vzniknou nové endpoint pro webhooks (.../*webhooks*), kde bude pro každou službu, která webhooks podporuje, vystaven vlastní endpoint. Bude tak jednodušší poznat, ze které služby zpráva je. Na něj budou zasílány případné zprávy ze synchronizovaných služeb. Při přijetí zprávy vrstva Api zkontroluje, zda má uživatel nárok na okamžitou synchronizaci. Pokud ano, předá zprávu vrstvě Core, která požadavek zpracuje. Pokud uživatel nemá na okamžitou synchronizaci nárok, uloží Api zprávu do WebhookQueue, kde si ji časem vyzvedne Core.

Core tím pádem musí mít vystavené endpointy pro okamžité zpracování zprávy, opět bude mít každá služba vlastní. Vrstva Api řeší požadavek jako první kvůli tomu, že Core vůbec nepracuje s členstvími a pouze provádí synchronizaci. Je proto rozumné nechat toto rozdělení úkolů i v případě webhooks.



Obrázek 3.6 Zpracování příchozího webhook, vytvořeno pomocí [67], notace BPMN [68]



Obrázek 3.7 Postup v synchronizačním jobu s webhooks, vytvořeno pomocí [67], notace BPMN [68]

	Created	Updated	Deleted
Projekt	Ticket → Timer	Ticket → Timer	
Úkol	Ticket → Timer	Ticket → Timer	
Časový záznam	Ticket ↔ Timer		Ticket ↔ Timer

Tabulka 3.1 Reakce na webhooks při spojení projektového nástroje s nástrojem pro měření času

3.3.7 Reakce na webhook pro spojení Timer a Ticket

V této části popisuji změny, které vyvolá přijetí zprávy přes webhook při spojení mezi projektovým nástrojem a nástrojem pro měření času. Efekt nemusí být okamžitý, protože zpráva může být odložena do fronty a zpracována později.

Timer2Ticket bude reagovat na webhooks o vytvoření (Created), aktualizaci (Updated) a odstranění (Deleted). Dále v těchto zprávách může být informace buď o projektu, úkolu, nebo časovém záznamu. Ostatní obdržené zprávy budou ignorovány. Kdy bude Timer2Ticket reagovat na který webhook je znázorněno v tabulce 3.1.

Projekt

Timer2Ticket bude reagovat pouze na projekty z projektového nástroje. Projekty z nástrojů pro měření času totiž nejsou do projektového nástroje přenášeny. Následující situace tedy popisují pouze projekty z projektového nástroje.

Created – Timer2Ticket musí nejdříve vytvořit obraz projektu v nástroji pro měření času. Po jeho vytvoření vytvoří v rámci spojení nové mapování objektů.

Updated – Jediné aktualizované pole, na které bude TImer2Ticekt reagovat, bude jméno projektu. Nejdříve se Timer2Ticket pokusí najít příslušné mapování a s tím i obraz ve druhém nástroji, pokud ho nalezne, změní ve druhé službě jméno projektu. V případě neúspěchu se nestane nic.

Deleted – Webhook pro smazání projektu nebude Timer2Ticket zpracovávat, protože mazání projektů neprobíhá okamžitě. Smazání projektu bude provedeno při příští synchronizaci konfiguračních objektů.

Úkol

Timer2Ticket bude reagovat pouze na úkoly z projektového nástroje. Úkoly (tagy) z nástroje pro měření času budu ignorovány, protože ani nyní nejsou úkoly přenášeny směrem z měřiče času do projektového nástroje. Následující situace tedy popisují pouze úkoly z projektového nástroje.

Created – Timer2Ticket musí nejdříve vytvořit příslušný úkol (tag) v nástroji pro měření času. Po jeho vytvoření vytvoří v rámci spojení nové mapování objektů.

Updated – Jediné zatím aktualizované pole bude jméno úkolu, změny jiných atributů úkolu nejsou pro Timer2Ticket v tomto kontextu důležité. Nejdříve se Timer2Ticket pokusí najít příslušné mapování a s tím i obraz ve druhém nástroji, pokud ho nalezne, změní ve druhé službě jméno úkolu. V případě neúspěchu se nestane nic (pokud měl úkol existovat, bude vytvořen při řádné synchronizaci).

Deleted – Webhook pro smazání úkolu nebude Timer2Ticket zpracovávat, protože mazání úkolů neprobíhá okamžitě. Smazání úkolu bude provedeno při příští synchronizaci konfiguračních objektů.

Časový záznam

U Časových záznamů musí Timer2Ticket reagovat na zprávy jak z projektového nástroje, tak z nástroje pro měření času.

Created – Při vytvoření nového časového záznamu musí dojít k vytvoření ekvivalentu ve druhé službě. Následně vznikne nové TESO a objekty STEO odkazujícími na objekty ve službách.

Updated – Nyní Timer2Ticket na změny v časových záznamech nereaguje a pro webhooks to bude stejné.

Deleted – Při smazání časového záznamu v původní službě dojde ke smazání i ve druhé službě. Následně musí být z databáze smazáno TESO a příslušná STEO. Pokud dojde ke smazání ve službě, která není primární, neudělá Timer2Ticket nic, ale při další synchronizaci bude tento objekt znovu vytvořen.

3.3.8 Reakce na webhook pro spojení dvou Ticket nástrojů

Následující část popisuje změny, které musí proběhnout při obdržení zprávy přes webhook pro spojení dvou projektových nástrojů. Změny nemusí být aplikovány okamžitě, ale mohou být uchovány ve frontě pro pozdější zpracování.

Projekt

Při spojení dvou projektových nástrojů jsou projekty k synchronizaci vybírány explicitně ve webovém uživatelském rozhraní. Webhooks pro projekty tedy nedává z následujících důvodů smysl zpracovávat.

Created – Nově vytvořený projekt nemohl být vybrán k přenášení časových záznamů. Projekt se ale uživateli zobrazí v nastavení synchronizace ve webovém rozhraní.

Updated – Timer2Ticket nepřenáší žádná data o atributech projektu mezi systémy.

Deleted – Smazání projektu v Timer2Ticket probíhá s časovým odstupem, nevadí tak čekat na řádnou synchronizaci.

Úkol

Timer2Ticket zajímají pouze úkoly z projektů, mezi nimiž je zapnutá synchronizace. Pokud je úkol z jiného projektu, tak žádné zpracování neproběhne.

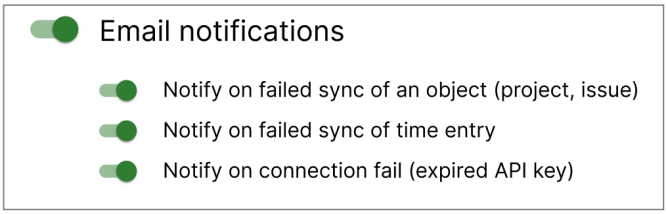
Created – Timer2Ticket zkontroluje zvolené synchronizační vlastní pole, pokud bude vyplněné, pokusí se vytvořit mapování.

Updated – Timer2Ticket zkontroluje zvolené synchronizační vlastní pole, pokud bude vyplněné, pokusí se Timer2Ticket vytvořit mapování.

Deleted – Na smazání nebude Timer2Ticket reagovat. Případné mazání proběhne až při řádné synchronizaci.

Časový záznam

Timer2Ticket zajímají pouze časové záznamy z linkovaných úkolů. Pokud je časový záznam z jiného úkolu, tak žádné zpracování neproběhne.



Obrázek 3.8 Nastavení e-mailových notifikací, vytvořeno pomocí [69]

Created – Timer2Ticket vytvoří obraz časového záznamu ve druhém systému a následně vytvoří TESO.

Updated – Aktualizaci časového záznamu Timer2Ticket nyní nezpracovává a bude tomu tak i u webhooks.

Deleted – Pokud byl časový záznam smazán ve zdrojovém systému, bude smazán i jeho obraz. Pokud byl smazán obraz ve druhém systému, bude znovu vytvořen.

3.3.9 Změny v uživatelském rozhraní pro nastavení webhooks

Do konfigurace frekvence spojení přibude možnost využívat na spojení webhook. Zároveň zde bude k dispozici návod, jak webhooks nastavit pro synchronizované nástroje. Pro Toggl Track to bude pouze informace o tom, že webhook bude nastaven nástrojem Timer2Ticket. Pro Jira zde bude návod, jak webhook zprovoznit přímo v Jira, a na jaké URL má Jira zprávy posílat. Za Redmine zde bude informace, že webhooks nejsou zatím tímto nástrojem podporovány.

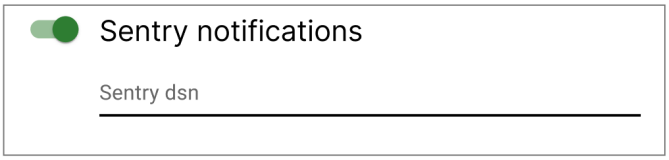
3.4 Uživatelské notifikace

Uživatel by do nástroje Timer2Ticket měl chodit ideálně co nejméně. Proto je důležité, aby existoval způsob, jak ho informovat o případných problémech při synchronizaci jinak, než ve webovém rozhraní. Z požadavků (NOT01 a NOT02) vychází 2 možné komunikační kanály: e-mail a Sentry.io. Uživatel také dostane možnost nastavit, které notifikace chce dostávat.

3.4.1 E-mailové notifikace

E-mailové notifikace budou ve výchozím stavu zapnuté. Pouze v případě, že by Timer2Ticket neznal e-mailovou dresu uživatele, budou notifikace vypnuté a pro jejich zapnutí bude nutné e-mailovou adresu vložit. Uživatel bude mít možnost e-mailové notifikace zrušit ve webovém rozhraní. Dále bude mít možnost omezit kategorie notifikací, které chce dostávat. Návrh je možné si prohlédnout na obrázku 3.8.

- Neprovedená synchronizace konfiguračního objektu (projekt, úkol)
- Neprovedená synchronizace časového záznamu
- Chyba ve spojení (neplatný API klíč nebo API point)



Obrázek 3.9 Nastavení Sentry.io notifikací, vytvořeno pomocí [69]

3.4.2 Notifikace přes Sentry

Pro zprovoznění hlášení chyb přes Sentry je nutné získat od uživatele DSN (webové URL) jeho Sentry projektu, kam chce zprávy dostávat. To mu bude umožněno v uživatelském nastavení notifikací. Přes Sentry budou následně posílány všechny zprávy, protože si je uživatel může filtrovat a zpracovávat sám lépe přímo v Sentry [58]. Návrh nastavení Sentry notifikací je k dispozici na obrázku 3.9.

3.4.3 Webové rozhraní pro nastavení notifikací

Nastavení notifikací vznikne jako nová záložka v nastavení profilu ve webovém rozhraní Timer2Ticket. Bude možné v něm pro jednotlivé kanály nastavit, zda mají být aktivní či nikoliv, případně jaký typ zpráv má být uživateli zasílán.

3.4.4 Zasílání notifikací

Zde popisuji několik způsobů jak do nástroje Timer2Ticket vložit samotné zasílání notifikací od nejjednoduššího řešení po nejvíc komplexní, ale flexibilnější a udržitelnější.

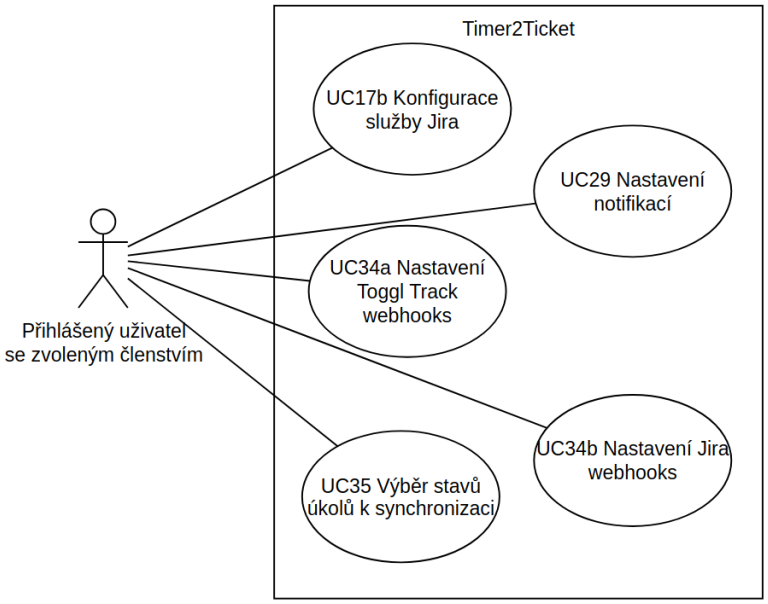
Notifikační třída v rámci vrstvy Core

V rámci vrstvy Core vznikne nová třída, které bude mít uživatelské notifikace na starost. Při kterékoliv chybě při jobu zavolá job metodu *notify(userId, error)* z této třídy. Ta se podívá do databáze, zjistí uživatelská nastavení a podle toho notifikuje uživatele příslušným kanálem. Implementace takové třídy bude relativně přímočará, ale například nebude možné zaslat notifikaci znovu v případě, že se ji nepodaří odeslat na první pokus. Také je pravděpodobné, že se notifikováním zpomalí provádění jobu.

Job pro zpracování notifikací

Sofistikovanější možností by bylo přidat notifikační job do Timer2Ticket Core. Ten by se spouštěl v pravidelných intervalech a k získání dat potřebných pro poslání notifikace by mu stačil záznam jobu v kolekci *JobLog*. Tam se totiž nachází jak id uživatele, které by notifikační job potřeboval k získání uživatelových nastavení notifikací, i jednotlivé chyby, které při jobu vznikly. Job by tak procházel předchozí logy a vytahoval z nich chybové hlášky, které by následně posílal správným kanálem.

Notifikace z jobu by nebyly odesílány okamžitě, ale vždy by měly alespoň drobné zpoždění. Vzhledem k povaze nástroje Timer2Ticket to nevnímám jako problém. Job by ale mohl poslat jednu notifikaci pro více chyb. Uživatel by tak obdržel pouze jednu notifikaci e-mailem za nějaké období a nestalo by se mu, že bude nástrojem Timer2Ticket spamován. Frekvence notifikací by navíc mohla být nastavitelná v uživatelském rozhraní (1 za hodinu, 1 denně, ...).



Obrázek 3.10 Případy užití, vytvořeno pomocí [59], notace UML [75]

Samostatná služba

Nejsložitější, ale podle mě nejlépe do budoucna udržitelnou variantou by byla samostatná služba starající se o notifikace. Core by tak stále řešilo pouze to, k čemu bylo původně navrženo a notifikace by byly spravovány v samostatné aplikaci. Ta by buď mohla dostávat požadavky přes API, nebo by mohla (podobně jako notifikační job) získávat údaje z logu jobů. Tato varianta by určitě mohla dělat vše stejně dobře jako notifikační job, ale z pohledu udržitelnosti a rozšiřitelnosti se rozhodně jedná o lepší variantu.

3.5 Případy užití

Případy užití zůstávají většinou stejné, jako byly popsány Jakubem Čermákem [4] a Martinem Paulem [5]. V této práci proto uvádím pouze ty případy užití, které bylo nutné aktualizovat, nebo se s nimi v předchozích pracích nepočítalo. Nové a aktualizované případy užití jsou znázorněny modelem na obrázku 3.10.

UC17b Konfigurace služby Jira

Tento případ užití navazuje na UC17 z práce Jakuba Čermáka [4]. Ten popisuje konfiguraci služby Redmine. Zde uvádím podrobné kroky pro podobnou konfiguraci Jira.

Aktéři: Přihlášený uživatel se zvoleným členstvím.

Počáteční podmínky: Téměř shodné s původním UC17. Uživatel se nachází na stránce konfigurace spojení, pouze má místo služby Redmine vybranou službu Jira.

Základní scénář:

1. Uživatel vyplní API klíč, Jira doménu a uživatelský e-mail.
2. Systém ověří korektnost spojení.

3. Systém informuje uživatele o úspěchu a uloží vyplněné přístupové údaje.
4. Uživatel zvolí, jakým způsobem chce nakládat s časovými záznamy bez přiřazeného úkolu.

Exception scénář: Tento scénář se spustí ve chvíli, kdy nastane problém s ověřením spojení ve druhém kroku základního scénáře.

1. Systém uživatele informuje, že ověření spojení neproběhlo korektně kvůli chybně zadanému API klíči, Jira doméně, nebo e-mailu.
2. Uživatel pokračuje prvním krokem základního scénáře.

UC29 Nastavení notifikací

Tento případ užití upřesňuje UC29 z práce Jakuba Čermáka [4]. Uživateli umožňuje zvolit notifikační kanál, vybrat skupiny zpráv a nastavit frekvenci notifikací.

Aktéři: Přihlášený uživatel se zvoleným členstvím.

Počáteční podmínky: Uživatel se nachází na stránce „Nastavení notifikací“.

Základní scénář:

1. Uživatel si vybere notifikační kanál a nastaví frekvenci notifikací.
2. Systém uloží nastavení a informuje uživatele o úspěchu operace.

Alternativní scénář: Tento scénář nastává v případě, že nedojde k úspěšnému uložení nového nastavení ve druhém kroku základního scénáře.

1. Systém informuje uživatele o chybě při ukládání nastavení.
2. Uživatel pokračuje prvním bodem základního scénáře.

UC34a Nastavení Toggl Track webhooks

Tento případ užití umožní uživateli nastavení webhooks pro Toggl Track. Díky webhooks bude Timer2Ticket schopen reagovat na změny v Toggl Track okamžitě a uživatel nebude muset čekat na pravidelnou synchronizaci, aby viděl naměřený čas i v projektovém nástroji.

Aktéři: Přihlášený uživatel se zvoleným členstvím.

Počáteční podmínky: Uživatel se nachází na stránce „Nastavení Webhooks“.

Základní scénář:

1. Uživatel povolí/zakáže aplikaci Timer2Ticket využívat webhooks pro Toggl Track.
2. Systém informuje uživatele o úspěchu či neúspěchu uložení nastavení.

UC34b Nastavení Jira webhooks

Tento případ užití umožní uživateli nastavení webhooks pro Jira. Díky webhooks bude Timer2Ticket schopen reagovat na změny v Jira okamžitě a uživatel nebude muset čekat na pravidelnou synchronizaci, aby viděl například nové úkoly i v nástroji pro měření času.

Aktéři: Přihlášený uživatel se zvoleným členstvím.

Počáteční podmínky: Uživatel se nachází na stránce „Nastavení Webhooks“.

Základní scénář:

Uživatel povolí/zakáže aplikaci Timer2Ticket využívat webhooks pro Jira.

Systém informuje uživatele o úspěchu či neúspěchu uložení nastavení.

Systém nabídne uživateli návod, jak webhooks zprovoznit v Jira.

Uživatel bude postupovat podle návodu a povolí webhooks pro aplikaci Timer2Ticket v Jira.

UC35 Výběr stavů úkolů k synchronizaci

Tento případ užití umožní uživateli v rámci každého spojení nastavit, které stavy úkolů mají být synchronizovány. Uživatel tak například může Timer2Ticket přikázat, aby ignoroval hotové úkoly. Ty pak nebudou zdržovat synchronizaci a z nástroje pro měření času budou odstraněny příslušné tagy.

Aktéři: Přihlášený uživatel se zvoleným členstvím.

Počáteční podmínky: Uživatel má nakonfigurované minimálně jedno spojení a nachází se na stránce tohoto spojení.

Základní scénář:

1. Uživatel označí/zruší označení stavu k synchronizaci.
2. Systém změní barvu tohoto stavu a aktualizuje seznam synchronizovaných stavů.

Alternativní scénář: Tento scénář nastává v případě, že aplikace není schopna uložit nový výběr synchronizovaných stavů ve druhém bodě základního scénáře.

1. Systém upozorní uživatele, že aktualizace stavů neproběhla správně.
2. Uživatel pokračuje prvním bodem základního scénáře.

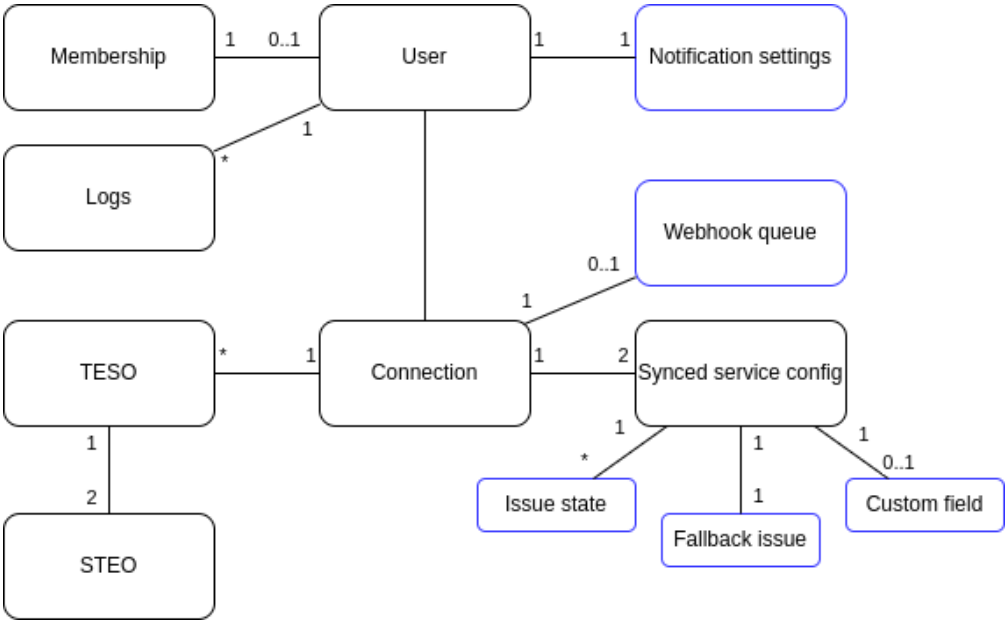
3.6 Metodika vývoje

Požadavky na rozšíření nástroje Timer2Ticket sesbírané v kapitole 2 lze rozdělit na čtyři [3] relativně nezávislé části:

1. Přidání Jira
2. Synchronizace dvou projektových nástrojů
3. Podpora webhooks
4. Notifikace

Každou z těchto částí budu implementovat jako funkční celek v rámci jednoho Scrum sprintu [76]. Výsledkem každého sprintu bude otestovaná a do produkce nasaditelná implementace. Připouštím, že požadavky z kapitoly 2 nejsou dokonale přesné, a proto budu v průběhu implementace v kontaktu se stakeholdery pro dosažení co nejlepšího možného výsledku.

[3] S „Enterprise“ verzí již nepočítám



Obrázek 3.11 Konceptuální datový model, vytvořeno pomocí [59], bez notace

3.7 Konceptuální datový model

Konceptuální datový model jsem vytvořil pro lepší pochopení souvislostí mezi jednotlivými daty, která v databázi budou částečně odpovídat jednotlivým kolekcím a v nich vnořeným objektům. Na modelu 3.11 jsou tato data znázorněna a barevně jsou odlišeny původní (černé) a nové (modré) entity. Detailní datový model je k dispozici v následující sekci 3.8.

3.7.1 Původní datové entity

Tyto entity se objevily již v předchozích pracích, konkrétně navazuji především na databázový model Martina Paula [5]. Ve svém modelu (obrázek 3.11) zmiňuji pouze podstatné věci pro tuto práci, proto vědomě zanedbávám například složitější strukturu logů.

User: Uživatel.

Membership: Členství uživatele.

Logs: Logy proběhlých jobů.

Connection: Spojení mezi synchronizovanými službami.

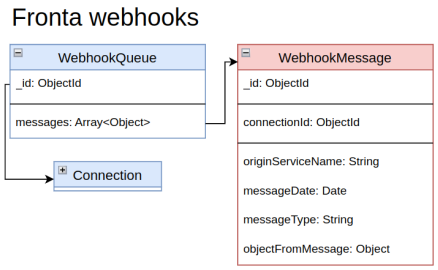
TESO: Time Entry Synced Object.

STEO: Service Time Entry Object.

Synced service config: Konfigurace synchronizované služby.

3.7.2 Nové datové entity

Přidáním služby Jira, implementací notifikací a webhooks vznikají nová data, se kterými bude Timer2Ticket pracovat. Kvůli Jira dojde k rozšíření o výchozí úkol, seznam stavů úkolů, která



Obrázek 3.12 Nová kolekce „Fronta Webhooks“, vytvořeno pomocí [59], bez notace

mají být součástí synchronizace a o přidání vlastního mapovacího pole. Spojení bude rozšířeno o webhooks a uživatel dostane možnost nastavení notifikací.

Issue state: Stavy úkolů, které budou využity pro omezení množství synchronizovaných konfiguračních objektů (úkolů-tagů) mezi službami.

Fallback issue: Výchozí úkol, pod který budou spadat v Jira časové záznamy, které nebudou mít přiřazený úkol.

Custom Field: Vlastní pole, podle kterého bude probíhat párování úkolů při spojení dvou projektových nástrojů.

Webhook queue: Fronta nezpracovaných zpráv obdržených přes webhooks. Pro jiná členství než Senior nebudou webhooks zpracovávány okamžitě, ale budou odkládány do fronty.

Notification settings: Nastavení notifikací uživatele, bude mít možnost zvolit notifikační kanál a frekvenci doručování zpráv.

3.8 Aktualizace databázového schématu

Většina databázového schématu zůstane shodná s modelem z kapitoly 3.6 diplomové práce Martina Paula [5], model jsem zde již uváděl a je k nahlédnutí na obrázku 2.5. Při návrhu na rozšíření tohoto modelu budu vycházet ze stejného schématu a použiji stejné modelovací konvence. Návrh rozšířeného modelu je znázorněn na obrázku 3.15. Nově přidané atributy jsou modrou barvou.

3.8.1 Nová kolekce WebhookQueue

Kolekce WebhookQueue bude sloužit pro nezpracované webhooks a bude obsahovat frontu nezpracovaných zpráv obdržených přes webhook endpointy (viz sekce 3.3.5). Každá zpráva ve frontě bude obsahovat především zdrojovou službu, objekt obsažený ve zprávě, jaká akce s ním byla provedena (created, updated, deleted) a datum změny objektu. Dále bude obsahovat referenci na spojení, ke kterému se zpráva váže (dodá API při vytváření). Nová kolekce WebhookQueue je znázorněna na obrázku 3.12.

3.8.2 Rozšíření kolekce Connection

Kolekce Connection bude rozšířena o několik atributů přímo na kořenové úrovni i hlouběji ve struktuře. Změny jsou způsobeny přidáním projektového nástroje Jira, přidáním spojení mezi dvěma projektovými nástroji a přidáním webhooks. Upravené schéma kolekce Connection je znázorněno na obrázku 3.13.

Obrázek 3.13 Upravená kolekce „Spojení“, vytvořeno pomocí [59], bez notace

Nové atributy v kolekci

Přímo na nejvyšší úrovni kolekce Connection přibudou nové atributy „projectMappings“ a „userMappings“. U obou se jedná o kolekce dalších objektů a budou potřeba při spojení dvou projektových nástrojů.

Synced Service

Do objektu Synced Service bude navíc přidán atribut „webhookOn“ indikující, zda je na dané službě aktivní webhook. Tohoto atributu bude využívat job procházející frontu webhookQueue.

ServiceConfig

Konfigurace služby je doplněna o atributy, které potřebuje Jira (doména a způsob naložení s časem k projektu - fallbackIssue). Dále bude přidáno pole pro mapovací custom field při spojení dvou projektových nástrojů „customField“ a pole „ignoreIssueStates“, které bude obsahovat seznam stavů úkolů, které mají být ignorovány při synchronizaci. Negace stavů úkolů je zvolena proto, že při přidání nového stavu úkolů do projektu bude takový stav synchronizován bez toho, aby bylo potřeba kontrolovat nové stavy. Dále bude změněn atribut userId (odkazující na id uživatele v synchronizované službě) z typu *number* na typ *number* nebo *string*, protože Jira má Id uživatelů vedena jako řetězce znaků.

ProjectMappings

Mapování projektů pro spojení dvou projektových nástrojů obsahuje pouze id projektů z obou nástrojů tak, aby bylo možné vytvořit párování.

UserMappings

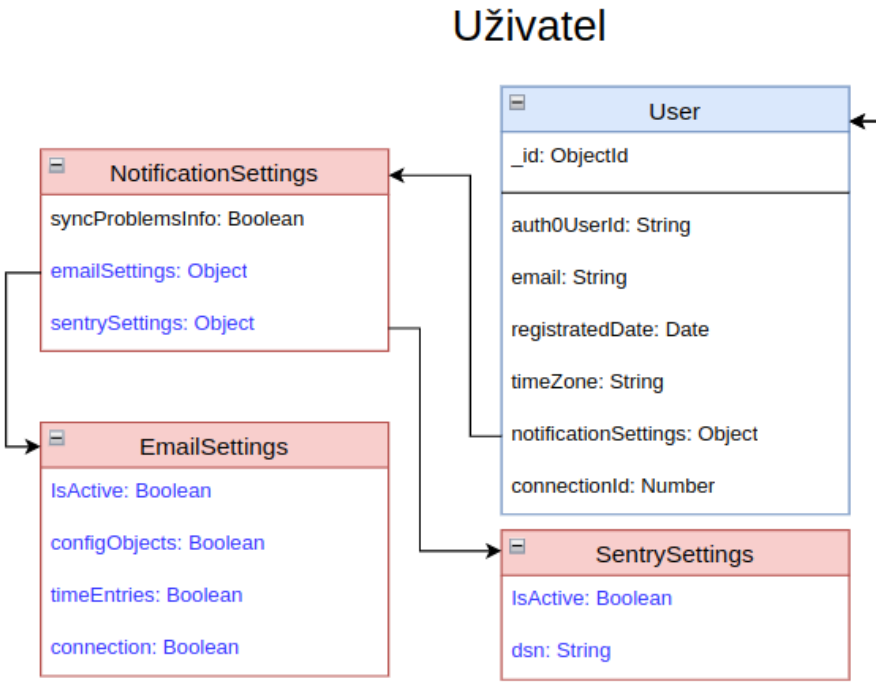
Mapování uživatelů bude obsahovat pár objektů uživatelů, z každé služby jednoho. Každý uživatel musí mít id a uživatelé z Jira musí navíc mít uložený API klíč a přihlašovací e-mail.

3.8.3 Rozšíření kolekce JobLog

Kolekce JobLog bude upravena v případě, že bude využita varianta notifikací s notifikačním jobem, nebo přes separátní službu, viz 3.4.4. Do kolekce bude v takovém případě přidán atribut „notificationHandled“, který bude sloužit pro notifikační job, který díky ní bude vědět, zda už tento log v rámci notifikací vyřídil.

3.8.4 Rozšíření kolekce User

Kolekce User bude rozšířena o notifikační nastavení. Zvlášť zde bude nastavení pro e-mailové notifikace a pro notifikace přes Sentry. Kolekce uživatele je znázorněna na obrázku 3.14



Obrázek 3.14 Upravená kolekce „Uživatel“, vytvořeno pomocí [59], bez notace

Obrázek 3.15 Schéma databáze, navazuje na kapitolu 3.6 práce Martina Paula [5], vytvořeno pomocí [59], bez notace

Kapitola 4
Implementace

V této kapitole popisuji, jakým způsobem jsem realizoval změny v aplikaci Timer2Ticket. Změny se dotkly databáze i vrstev Core, Api a webového rozhraní aplikace. Detailně zde vysvětluji realizaci jednotlivých požadavků podle analýzy i návrhu. Samozřejmě, že oproti návrhu došlo v realizaci ke změnám. Jejich důvody i řešení také diskutuji v této kapitole.

4.1 Přidání nástroje Jira

4.1.1 Spojení Jira a Toggl Track

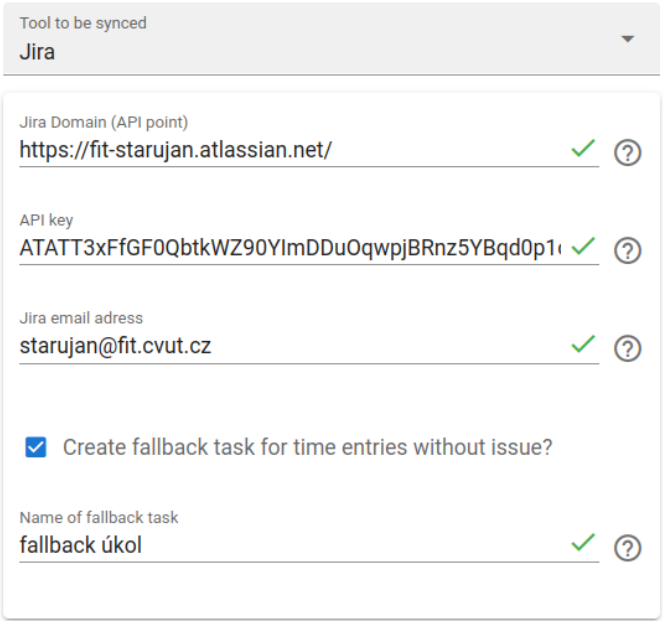
Při implementaci spojení Projektového nástroje Jira a Toggl Track jsem postupoval podle návodů pro přidání další služby od Martina Paula [5] a od Víta Štefana [1]. Implementace Jira do nástroje Timer2Ticket se zásadně neliší od implementace předchozích nástrojů, proto zde budu uvádět pouze části, které se vymykají předchozímu chování popsanému v pracích [1, 4, 5].

4.1.2 Synchronizace času naměřeného k projektu

Při implementaci synchronizace času vykázaného k projektu jsem se mírně odchýlil od původního návrhu z kapitoly 3.1.7. V uživatelském rozhraní jsem rozdělil nastavení do dvou samostatných částí, kde první je checkbox, který určuje, zda má být takový záznam vůbec synchronizován. Druhou částí, která je zpřístupněna pouze v případě, že je checkbox zaškrtnutý, je textové pole, kam uživatel napíše jméno úkolu, do kterého chce časové záznamy bez úkolu synchronizovat (jméno úkolu musí být neprázdné). Výsledný vzhled formuláře je zachycen na obrázku 4.1.

Pokud uživatel synchronizovat nechce, časový záznam z Toggl Track bude při synchronizaci ignorován a do Jira se nepropíše. Pokud ale je zvolen úkol, tak se Timer2Ticket nejdříve v příslušném projektu pokusí tento úkol najít přes JQL query [70]. Hledání probíhá přes název úkolu (v Jira atribut *summary*). Při nalezení přesné shody bude čas synchronizován s tímto úkolem. Pokud ale úkol nalezen, Timer2Ticket takový vytvoří a pak do něj vloží časový záznam.

V původním návrhu jsem počítal s tím, že budu někam ukládat informaci o id takového úkolu (v každém projektu). Hledáním úkolu přes query mi tato starost odpadá, ale má to několik dopadů pro uživatele. Za pozitivní považuji ten, že uživatel má takto možnost vybrat už existující úkol z nějakého projektu. Protože pokud zadá jméno existujícího úkolu, tak Timer2Ticket ho nalezne a nebude vytvářet nový. Problém ale nastává při neunikátnosti jména úkolu v rámci projektu. Timer2Ticket pak není schopen říct, do kterého z nich záznam patří a proto přidá k prvnímu nalezenému. K duplicitě jména úkolu může dojít pouze v případě, že někdo jiný vytvoří úkol se stejným jménem později než Timer2Ticket (Timer2Ticket už má vytvořené mapování a neovlivní



Obrázek 4.1 Finální implementace konfigurace spojení Jira

ho to), nebo že v době konfigurace synchronizace spojení již existují v Jira 2 úkoly se stejným jménem. V takovém případě Timer2Ticket opravdu nedokáže rozhodnout, který z úkolů je ten správný a vybere jeden z nich.

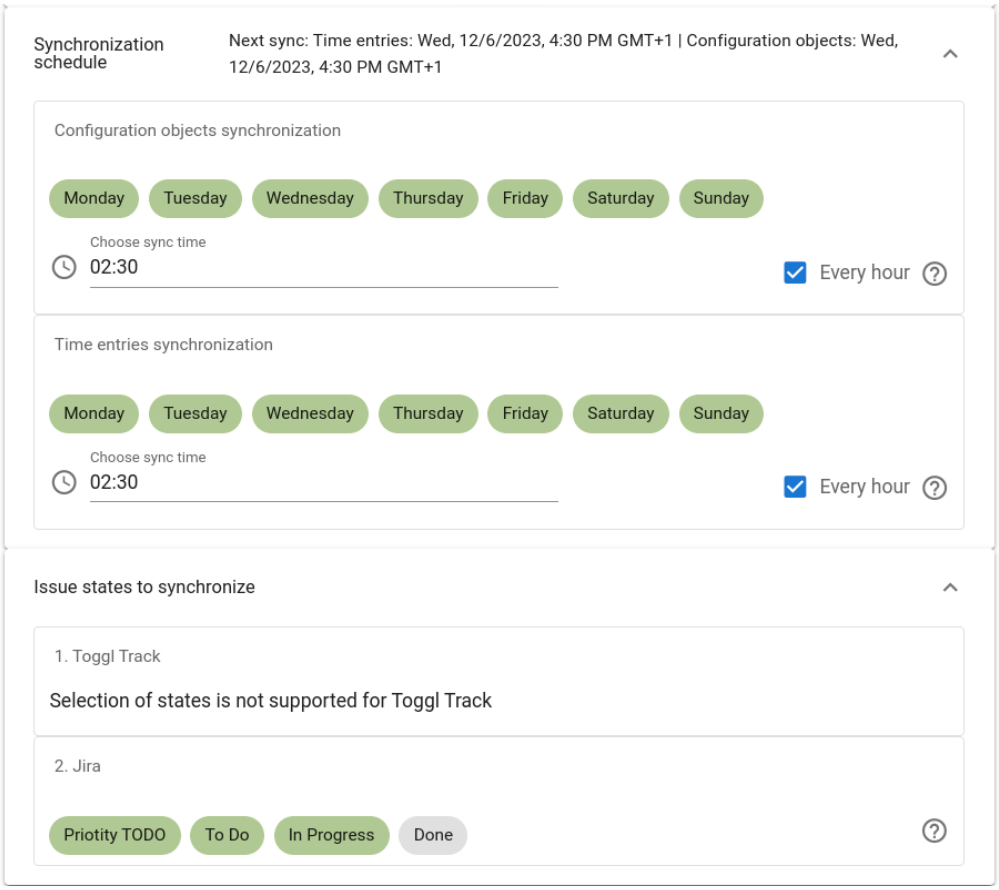
Dalším pozitivním výsledkem této implementace je, že více uživatelů, kteří si nakonfigurují toto spojení, může vybrat stejný „Fallback“ úkol. Pokud tedy já a můj kolega pracujeme každý s vlastním Toggl Track, ale společnou Jira, můžeme se domluvit na jednom „Fallback“ úkolu a používat ho oba. Pokud dojde v průběhu života spojení ke změně jména „Fallback“ úkolu (v parametrech spojení), tak Timer2Ticket bude hledat úkol s novým jménem a bude časové záznamy k projektu synchronizovat tam. Staré záznamy už zůstanou přiřazené původnímu úkolu a nové budou synchronizovány k novému. Nepředpokládá se, že by uživatel měnil nastavení spojení často a proto mi tato vlastnost přijde z uživatelského pohledu rozumná a dobře pochopitelná.

4.1.3 Frontend

V rámci implementace Jira ve webovém rozhraní byla vytvořené komponenta *ToolSetUpper.vue*. Uživatel má díky této komponentě k dispozici formulář pro vložení přihlašovacích údajů. Formulář je velmi podobný formuláři pro Toggl Track a Redmine.

Umístění filtrování úkolů podle stavu

Při implementaci výběru stavů ve webovém rozhraní jsem se oproti původnímu návrhu 3.2 rozhodl odstranit výběr stavů z konfigurace spojení a vytvořit pro něj samostatnou část podobně, jako to má nastavení rozvrhu synchronizace spojení (viz horní část obrázku 4.2). Výběr stavů totiž není v rámci konfigurace spojení nezbytně nutný. Uživatele nijak nepoškodí, pokud vybere stavy úkolů, které synchronizovat nechce, později. Odebráním této možnosti z konfigurace spojení tak zůstanou v konfiguraci spojení pouze povinné položky. Navíc je takto usnadněná případná změna modelu předplatného, kdy je možné jednodušeji tuto funkcionalitu schovat za některé z předplatných.



Obrázek 4.2 Plánování rozvrhu synchronizace a výběr stavů k synchronizaci

Způsob výběru ale zůstal shodný s návrhem. Stále jsou k dispozici čipy, které jsou ve výchozím stavu všechny vybrané a uživatel má možnost jejich výběr zrušit a synchronizaci tohoto stavu tak odebrat. Po odebrání stavu ze synchronizace bude stále probíhat synchronizace časových záznamů u dříve vybraných stavů, ale úkoly s tímto stavem zmizí z nabídky tagů v Toggl Track. Synchronizace časových záznamů probíhá dál zejména kvůli plánování jobů. Pokud bych totiž okamžitě přestal časové záznamy synchronizovat, tak bych potenciálně okradl uživatele o časové záznamy vytvořené mezi poslední synchronizací a odebráním stavu ze synchronizovaných stavů. Nový výběr stavů k synchronizaci je zachycen na obrázku 4.2.

4.1.4 Api

Vrstvy Api se tato úprava dotkla převážně tak, že ukládá nové informace z webového rozhraní do databáze.

Filtrování úkolů podle stavu

Při výběru stavů k synchronizaci poskytuje pro frontend vrstva API všechny možné stavy úkolů. Oproti návrhu 3.1.7 ale nedostane uživatel k výběru všechny možné stavy v Jira, protože každý projekt v Jira má stavy vlastní. Uživatel by pak například vybíral z několika stavů *Done*, což by rozhodně nebylo přehledné. Proto jsem místo stavů zvolil atribut *statusCategory*, který určuje, do které kategorie stavů patří konkrétní stav (například *In progress*, nebo *Done*). Toto členění je

také v Jira konfigurovatelné a je o něco hrubší (více stavů sdílí jednu statusCategory), ale díky tomu zase pro uživatele přehlednější.

4.1.5 Core

Komunikace s Jira

Pro komunikaci s Jira slouží třída *JiraSyncedService*, která implementuje interface *SyncedService*. *Synced Service* definuje rozhraní pro veškerou komunikaci se službami třetích stran. Služby pro komunikaci s Redmine i Toggl Track také implementují stejné rozhraní a Jira se tak nijak neodchyluje od původního návrhu.

Vyhledávání úkolů

V kapitole návrh 3.1.9 jsem počítal s tím, že bude třeba pro získání všech časových záznamů obvolat postupně všechny úkoly v Jira. Po lepší analýze se ukázalo, že je možné rozšířit vyhledávání úkolů o časové záznamy. Tím pádem je třeba se dotazovat jednotlivých úkolů na jejich časové záznamy pouze v případě, že se všechny časové záznamy úkolu nevejdou na jednu stránku (je jich více než 10). Tím pádem je nutný výrazně menší počet API volání Jira při synchronizaci časových záznamů. Místo $O(n)$, kde n je počet úkolů se počet volání dostává řádově na jednotky ($O(1)$) API volání.

Na možnost lépe využívat a pracovat s objekty získanými přes API mě navedl na jedné z konzultací Oldřich Malec. Probírali jsme webhooks a díky tomuto vylepšení získávání dat z Jira ztratila na významu původně plánovaná implementace WebhookQueue 3.3.5. A to proto, že by ve výsledku neušetřila žádné volání, což byl hlavní důvod její existence. Více o implementaci webhooks v kapitole 4.3

Filtrování úkolů podle stavu

Vrstva Core při synchronizaci s Jira upravuje při vyhledávání úkolů query atributy tak, aby docházelo k filtrování úkolů již na straně Jira. Timer2Ticket tak ušetří velké množství dotazů na Jira API je v tomto velmi flexibilní a JQL pro filtrování funguje dobře. Filtrace probíhá přes atribut *statusCategory*.

Díky tomuto omezení se budou úkoly, které mají stav, který se nesynchronizuje, tvářit pro Timer2Ticket jako smazané a bude tak k nim i přistupovat. Mapování každého takového úkolu bude označeno, že má být smazáno. Až uplyne určitá doba (standardně sedm dní), tak bude odstraněno mapování a s ním i příslušný objekt (tag) v Toggl Track.

Řešení zahlcení Jira API

Jira API je schopno reagovat na omezený počet dotazů za určitou dobu (počet závisí na úrovni předplatného) [77]. Pro implementaci kontroly, zda není server zahlcen jsem vycházel z pseodokódu od Atlassian přímo v dokumentaci [77]. Implementace při obdržení odpovědi o zahlcení serveru zjistí z hlavičky odpovědi, jak dlouho má čekat s dalším dotazem. Intervaly mezi voláními se postupně prodlužují a po čtvrtém neúspěchu je volání prohlášeno za neúspěšné.

4.2 Spojení dvou projektových nástrojů

V rámci implementace spojení projektových nástrojů vycházím z funkčního rozhraní pro komunikaci s jednotlivými službami (Redmine a Jira). Postupoval jsem z velké části podle návrhu 3.2 v této sekci zmiňuji pouze odchylky od něj a změny provedené v kódu.

4.2.1 Synchronizace času přiřazeného k projektu

Pozorný čtenář již z předchozích kapitol ví, že v Redmine je možné vykazovat čas nejen k úkolům, ale i pouze k projektu, bez vybraného úkolu. Při synchronizaci by pro přenesení takového času bylo nutné vybrat nebo vytvořit úkol pro tento čas. Protože je ale synchronizace časových záznamů mezi projektovými nástroji prováděna pouze v případě, že je nalezeno spojení úkolů pomocí odkazu na úkol ve druhé službě, musel by se přístup k času k projektu lišit. V Jira by totiž musel existovat úkol, který by byl spojený s projektem v Redmine a pak by do tohoto Jira úkolu byly vkládány časy zaznamenané k projektu v Redmine.

Toto řešení je nepochybně implementovatelné, ale z mého pohledu nedává smysl takové časy do Jira vykazovat. Fungování synchronizace je nastaveno tak, že se synchronizuje pouze to, co uživatel explicitně označí, že se synchronizovat má. Navíc pokud funguje projekt najednou ve dvou nástrojích, mělo by jeho řízení (a s tím související zaznamenávání času) být v souladu s oběma nástroji a pro přenesení takového časového záznamu by bylo podle mě třeba výrazně poškodit workflow projektů v Jira. Proto jsou časové záznamy k projektu při synchronizaci časových záznamů mezi projektovými nástroji ignorovány.

4.2.2 Mapování úkolů 1:N

Implementace požadavku SPN04 (sekce 2.6.4), který požaduje, aby bylo možné jeden úkol z jednoho systému spojit s více úkoly v systému druhém. To možné je. Například: pokud bude více „malých“ úkolů v Redmine odpovídat jednomu „velkému“ úkolu v Jira, pak budou všechny časové záznamy z malých úkolů synchronizovány k Jira úkolu. Uživatel pak uvidí součet všech časových záznamů ze všech těchto malých úkolů z Redmine v jednom Jira úkolu.

I v případě vytvoření časového záznamu k úkolu, na který je navázáno více úkolů v systému druhém, dojde k synchronizaci tohoto záznamu. Timer2Ticket ale nemá v takovou chvíli definované, ke kterému z „malých“ úkolů tento časový záznam připíše. Vždy ale bude takový časový záznam pouze jeden, takže alespoň suma časů zůstane stejná. Schématicky je toto znázorněno na obrázku 4.2.2.

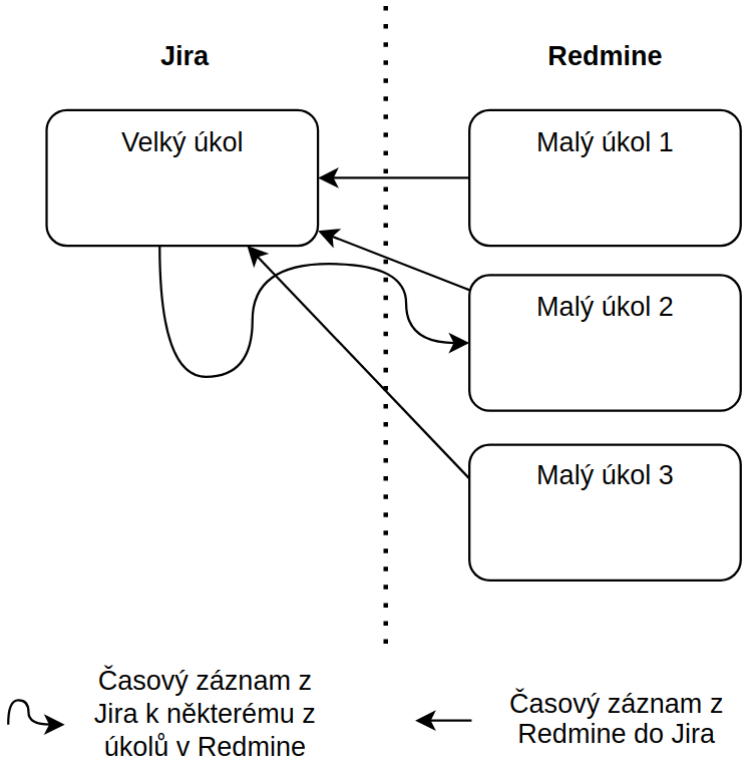
Toto chování je způsobeno částečně tím, že na tento use case nebyl kladen při implementaci důraz (pracoval jsem především se „skládáním“ časů k velkému úkolu) a také tím, že Timer2Ticket ve své implementaci původně nepočítal s takovýmto typem mapování. Proto bych nedoporučoval tuto funkcionalitu příliš využívat. Do budoucna ale určitě lze definovat chování v takovém případě a nástroj Timer2Ticket upravit.

4.2.3 Frontend

Rozšíření formuláře o výběr vlastních polí a párování projektů

Formulář konfigurace spojení byl rozšířen o výběr synchronizačního vlastního pole pro každou službu a následné párování projektů. Výběr vlastního pole definuje, podle čeho budou následně úkoly mezi službami mapovány. K výběru dostane uživatel při konfiguraci všechny vlastní pole, které Timer2Ticket u úkolů z dané služby našel. Vlastní pole bude možné měnit pouze do té doby, než dojde k prvnímu uložení spojení, ne později. Je to kvůli náročnosti implementace, protože v případě změny mapovacího pole by bylo nutné vymazat všechny časové záznamy vytvořené Timer2Ticket, vymazat všechny vazby mezi úkoly a následně vše začít znovu. Tento složitý proces nemusí být uživateli na první pohled patrný, proto to nebude povoleno. Navíc nepředpokládám, že by byl tento use case reálně využíván.

Po vybrání vlastního pole u obou služeb budou uživateli zobrazeny projekty z obou synchronizovaných nástrojů. Timer2Ticket nabídne k výběru ovšem pouze takové projekty, jejichž úkoly mají zvolené vlastní pole v nabídce. Protože pokud projekt toto vlastní pole neobsahuje, nemá smysl vůbec o synchronizaci uvažovat, protože nikdy nejde k požadovanému mapování. Když



Obrázek 4.3 Implementace synchronizace časových záznamů při mapování úkolů 1:N pro spojení dvou projektových nástrojů, vytvořeno pomocí [59], bez notace

uživatel nenajde projekt, který chtěl synchronizovat, tak má možnost do něj vybrané vlastní pole přidat (v projektovém nástroji) a v konfiguraci spojení znovu načíst projekty s daným vlastním polem.

Párování projektů pro spojení probíhá tak, že uživatel označí jeden projekt z první služby a jeden projekt ze druhé služby, následně přidá pár tlačítkem „ADD PAIR“ uprostřed, mezi boxy pro projekty. Pár projektů pak bude vytvořen a uložen, ale na rozdíl od vlastních polí, páry projektů lze přidávat a odebírat i po uložení spojení.

4.2.4 Api

Kvůli synchronizaci dvou projektových nástrojů je třeba získat ze synchronizovaných služeb všechna možná vlastní pole. U Jira se toto děje celkem přímočaře, Jira API poskytuje endpoint, přes který lze získat všechna možná vlastní pole napříč doménou a všechny projekty tato vlastní pole mají.

U Redmine je situace složitější. Vlastní pole existují jednak pro projekt (ty mě v tomto případě nezajímají) a zvlášť pro úkoly a nastavují se zvlášť v rámci každého projektu. K nastavení vlastních polí úkolů v jednotlivých projektech se ale nelze dostat přes žádný API endpoint [1]. Abych tedy získal možná vlastní pole úkolů, tak se nejdříve zeptám na úkoly, a následně z nich vyzobu jednotlivá možná vlastní pole. Rozlišuji při tom, do jakého projektu úkol patří a díky tomu jsem pak schopen dopočítat, jaká jsou pro každý projekt možná vlastní pole. Toho využívá webová aplikace při filtraci projektů pro zobrazení uživateli k párování.

Za každý projekt potřebuji načíst alespoň jeden úkol. Proto nastane problém, pokud v projektu není žádný otevřený úkol (při tomto dotazu pracuji pouze s otevřenými úkoly). V takovém případě to bude vypadat, že projekt žádná vlastní pole nemá a nebude tak nikdy uživateli nabídnut k párování na základě vybraného vlastního pole.

4.2.5 Core

Job synchronizace konfiguračních objektů

Ve vrstvě Core bylo třeba přizpůsobit job pro synchronizaci konfiguračních objektů. Ten má za úkol mapovat a synchronizovat mezi nástroji projekty a úkoly (kapitola 2.5 [1]). Původní kód počítal s primární a sekundární službou (Ticket primární a Timer sekundární) a vytvářel nové objekty v nástrojích. Proto jsem se rozhodl nezasahovat do stávajícího „Timer2Ticket“ jobu, ale v rámci jobu jsem vytvořil větev pro synchronizaci objektů „Ticket2Ticket“. Tato implementace pak pouze hledá páry na základě vybraného vlastního pole a vzájemného linkování úkolů v jednotlivých nástrojích. Žádné nové objekty nejsou vytvářeny jinde, než přímo v nástroji Timer2Ticket.

Při spuštění job nejdříve stáhne projekty a úkoly z obou služeb a pokusí se mezi nimi vytvořit mapování na základě vybraných vlastních polí a datech v nich. Pro slinkování objektů stačí mít informaci v jedné ze služeb (jen A má odkaz na B). Následně jsou nová mapování objektů uložena a stará (objekty byly odstraněny v původní službě) smazána z Timer2Ticket.

Problém s Jira id a Klíčem

Při kopírování id či odkazu na úkol se uživatel nedostane ve webovém rozhraní k id, ale ke klíči (key). Tento problém jsem již rozebíral v kapitole 3.1.3. Zbytek Timer2Ticket ale funguje s id, proto je třeba provést dotaz na Jira API (*...issues/issueKey*) a z odpovědi uložit id úkolu, nikoliv klíč. Toto řešení sice zpomaluje job, ale bez id by nebylo možné provést porovnání na shodu s objekty ve druhém nástroji.

[1] Technicky to lze, ale je k tomu třeba administrátorských oprávnění na celý Redmine. To by ale pro Timer2Ticket nemělo být třeba, proto tuto možnost neuvažuji. [66]

Samotné volání Jira není součástí kódu jobu, ale došlo k rozšíření třídy *JiraSyncedService* o metodu *getIssueIdFromIssueKey(key)*.

4.3 Webhooks

Zpracování webhooks bylo implementováno pro služby Jira a Toggl Track. Uživatelům s nejdražším členstvím Senior je nyní umožněna okamžitá synchronizace. Novému úkolu v Jira bude okamžitě vytvořen příslušný tag v Toggl Track a podobně.

Webhooks byly při implementaci vnímány jako doplněk ke stávající pravidelné synchronizaci a nemají ambici pravidelné synchronizace nahradit. Pokud se nepodaří webhook zpracovat, tak Timer2Ticket neřeší, jak by se případná chyba dala opravit. Místo toho webhook zahodí a počítá s napravením stavu při řádné synchronizaci.

Na webhooks se nelze spolehnout především proto, že nastavení musí (minimálně u Jira) provést uživatel a Timer2Ticket má velmi omezené možnosti, jak zjistit, zda byl webhook nakonfigurován správně. I z uživatelských testů 5.3 mi vyšlo, že správně nakonfigurovat webhook není samozřejmostí. A protože Timer2Ticket není registrován v ekosystému Jira, tak nemá možnost volat API entpoints, které by mu umožnily nastavovat, nebo alespoň zkoumat stav webhooks [62].

4.3.1 Scope jednoho webhook

Každý jednotlivý webhook (v každé službě je možné mít více konfigurovaných webhooks) je navázán na konkrétní spojení v Timer2Ticket. Pokud tedy má uživatel více spojení, u kterých by chtěl webhooks využívat, musí mít nakonfigurovaný každý webhook zvlášť. Jinak by bylo pro Timer2Ticket obtížné určit, ke kterému spojení nově vytvořený úkol (a nejen úkol) patří a kde všude má proběhnout aktualizace objektů.

Proto je v URL, na kterém jsou webhooks odchytávány i databázové id spojení, ke kterému Webhook patří. Obecně využívám *.../webhooks/{jméno služby}/{id spojení}*. Zpracování se totiž navíc liší podle volané služby (různé objekty služeb) a již v návrhu 3.3 jsem navrhl různé endpoints podle služeb a od tohoto návrhu jsem se neodchýlil. Id spojení je zvoleno jako identifikátor, protože frontend tuto informaci již má a komunikace s vrstvou Api byla i u dalších endpoints založena na id spojení. Navíc se podle id v databázi dobře hledá a vyhnul jsem se tak zavádění dalšího atributu.

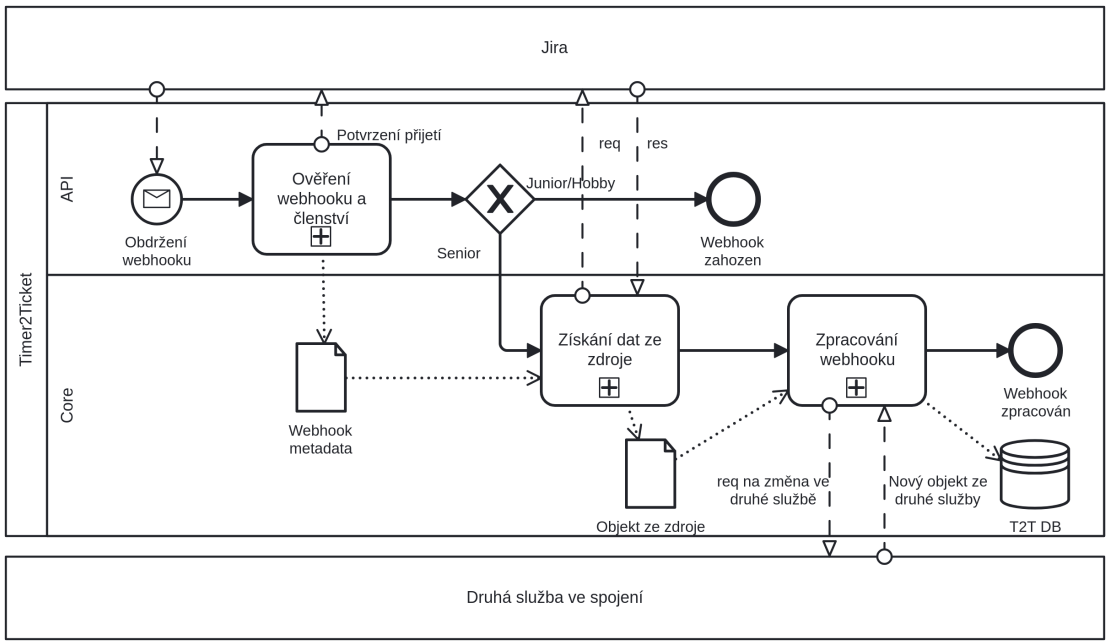
4.3.2 Průběh zpracování webhook

Webhook obdrží a jako první zpracuje vrstva Api. Ta nejdřív zkontroluje, zda má uživatel odpovídající členství a zda jsou základní atributy těla webhook v pořádku. Následně vytvoří objekt (univerzální pro všechny služby) pro vrstvu Core a tento objekt Core předá přes endpoint *POST .../webhooks*.

Core tento objekt přijme a ve třídě *WebhookHandler* se postará o zpracování webhook. Nejdříve zavolá službu, která webhook vytvořila a zjistí aktuální podobu objektu (proč toto volání probíhá vysvětluji v části „Bezpečnost“ 4.3.6). Následně pracuje s webhook podle jeho typu (projekt/úkol nebo časový záznam) a jaká událost (create, update, delete) webhook vyvolala. Zjednodušený proces zpracování webhook je zachycen na obrázku 4.4.

4.3.3 Problém s cyklením webhooks

Při zpracování webhook dojde k volání druhé služby v rámci spojení (pro vytvoření a aktualizaci dat). Pokud ale druhá služba má také v provozu webhooks, tak volání z Timer2Ticket bude zdrojem pro další webhook, který bude Timer2Ticket muset řešit. Pokud toto nebude ošetřeno,



Obrázek 4.4 Implementace přijetí a zpracování webhooks, vytvořeno pomocí [67], notace BPMN [68]

dojde k cyklickému volání služeb navzájem. To by přineslo chyby v obou službách. Při úvahách o ošetření tohoto chování jsem zkoumal tyto varianty:

Kontrolovat databázi Timer2Ticket a odmítat nedávno zpracované požadavky – Při přijetí každého webhook by Timer2Ticket kontroloval, jak dávno byly objekty, které souvisí s příchozím webhook, aktualizovány. U projektů a úkolů se jedná o mapování v rámci spojení, u časových záznamů je třeba vyhledat příslušné TESO. Lze pak například rozhodnout, že aktualizace proběhla před méně než 1 minutou a proto webhook nebude přijat. Tento přístup zároveň pojišťuje ignorování webhooks v průběhu řádné synchronizace a navíc i chvíli po ní.

Nevýhodou je prohledávání databáze při každém přijatém webhook.

Kontrolovat související objekt ve druhé službě – Tento návrh řešení by obnášel pro každý obdržený webhook zkontrolovat stav mapovaného objektu (pokud mapování existuje) ve druhé službě ve spojení. Přidává se tím navíc zbytečně synchronní volání druhé služby a zpracování webhooks by tak bylo výpočetně i časově náročnější.

Umožnit pouze jedné službě ve spojení používat webhooks – Pokud by uživatel měl ve webovém rozhraní možnost přepínat, u které ze služeb ve spojení budou webhooks přijímány, nemohlo by dojít k cyklení. Znamenalo by to ale úpravu uživatelské rozhraní i databáze o možnost přepínání, které ze služeb ve spojení má aktivní webhooks. To vyžaduje potenciálně častější zapojení uživatele. Ten by navíc přišel o půlku webhooks volání, které by asi nebylo z jeho pohledu dobře pochopitelné.

Nepřijímat a nezpracovávat webhooks, které by způsobovaly cyklení – Pokud by byly v implementaci ignorovány webhooks pro aktualizaci, a webhooks pro vytvoření nového objektu by byly správně kontrolovány, tak by k cyklení nedocházelo. Uživatel by tak ale přišel o možnost mít změnu propsanou okamžitě do druhé služby. Problém tohoto řešení je, že uživatel může například v Toggl Track zapnout stopky, změnit jméno, přiřadit tag a až potom měření vypnout. Každý z těchto úkonů vyvolá webhook, který informuje o aktualizaci. Uživatel by tak přišel i o velké množství okamžitých synchronizací, u kterých by

to nepředpokládal. Toto řešení je ale nejjednodušší na implementaci a nejméně náročné na výpočetní výkon.

Zvolená metoda zabránění cyklení webhooks

Nakonec jsem zvolil první zmíněnou metodu. Při přijetí webhook kontroluji, jak dávno byl objekt, kterého se webhook týká měněn a webhooks odmítám v případě, že je daný objekt upravován po méně než minutě. Pokud by v této době měl být webhook skutečně správně zpracován, tak dojde ke zpracování dat až při dalším synchronizačním jobu.

Jedna minuta je výsledkem vlastního testování, kdy webhooks z Jira chodí většinou prakticky okamžitě, ale webhooks z Toggl Track jsou často odeslány až po třeba deseti sekundách. Na více než deset sekund jsem nenarazil, proto bude potenciálně možné časem zvážit snížení času pro odmítnutí webhooks kvůli cyklení z jedné minuty na kratší čas.

Timeout je definován konstantou *webhooksAntiCycleTimeout* v souboru *constants.ts* ve vrstvě Api.

4.3.4 Nastavení webhooks v Jira

Jira neumožňuje aplikacím, které nejsou integrovány do jejich ekosystému používat API endpoints pro práci s webhooks [62]. Proto je nastavení na uživateli. Je mu proto k dispozici návod, jak webhooks nastavit a je na zodpovědnosti uživatele, aby to provedl správně a následně držel webhook v provozu. Timer2Ticket s tím bohužel neumí pomoct a je to jeden z hlavních důvodů, proč se nelze na webhooks stoprocentně spoléhat a nahradit jimi pravidelnou synchronizaci.

4.3.5 Nastavení webhooks v Toggl Track

Toggl Track API umožňuje veškerou potřebnou manipulaci s webhooks. Timer2Ticket tak po uživateli vyžaduje pouze souhlas s vytvořením webhook k danému spojení (automaticky to Timer2Ticket nedělá kvůli omezenému počtu možných webhooks v Toggl Track). Po udělení tohoto souhlasu Timer2Ticket webhook založí a zvaliduje (znázorněno na obrázku 4.5). Dál se o webhook nestará, ale pokud ho například uživatel smaže a chtěl by ho obnovit, tak v uživatelském rozhraní Timer2Ticket takovou možnost má.

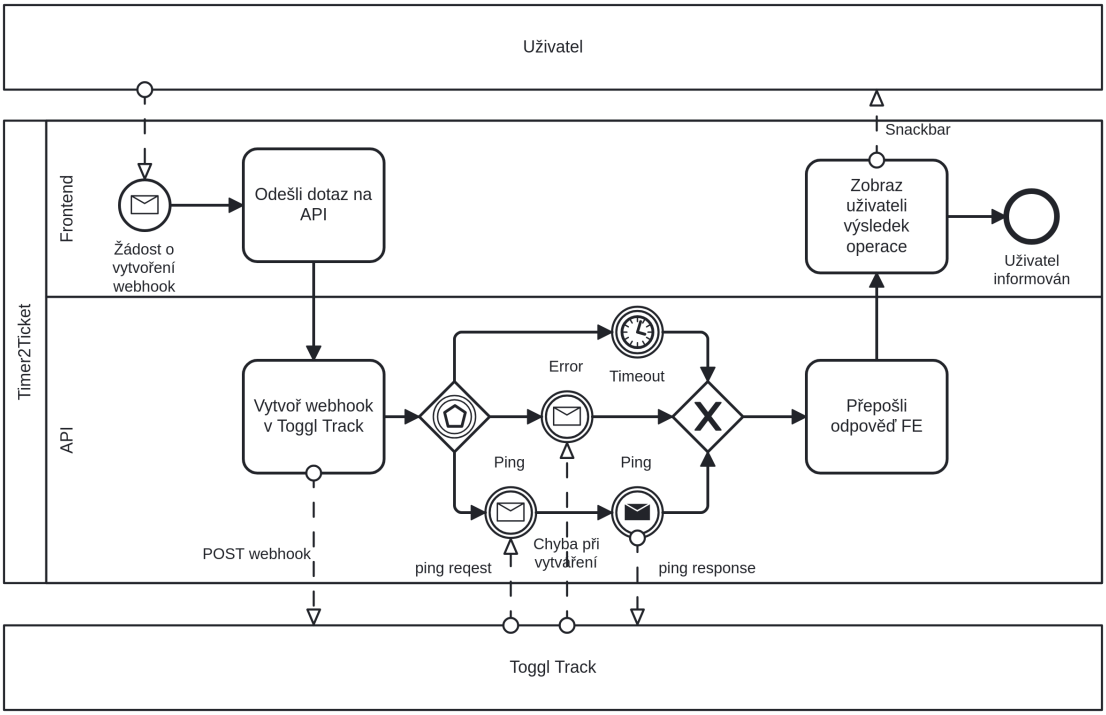
4.3.6 Bezpečnost

Pro rozpoznání, že webhook pochází z důvěryhodného zdroje nabízí jak Toggl Track, tak Jira možnost využití secret („hesla“), které je posláno spolu s webhook a díky tomu by Timer2Ticket dokázal ověřit volajícího. Pro využití secret by bylo ale v Timer2Ticket nutné vybudovat infrastrukturu pro jeho generování, předání uživateli a případné zneplatnění a vygenerování nového. Proto jsem se rozhodl secret nevyužívat a k validaci volajícího přistupuji jinak.

Při zpracování webhook ve vrstvě Core provedu GET request na vzdálený zdroj (podle id objektu z obdrženého webhook) stejně, jako se to děje při normální synchronizaci (bezpečnost zajištěna uživatelovými přihlašovacími údaji). Toto řešení výrazně zjednodušuje vnitřní implementaci Timer2Ticket a také klade menší nároky na uživatele při nastavování webhooks.

Díky tomuto přístupu je navíc výrazně usnadněna práce Timer2Ticket při odložení webhooks do fronty k pozdějšímu zpracování. Takto stačí uložit, že v objektu s nějakým id a typem (úkol, projekt, časový záznam) došlo ke změně, a proto je ho nutné zpracovat. Timer2Ticket tak bude zpracovávat pouze jednu změnu i v případě, že se daný objekt měnil vícekrát v době mezi pravidelnými joby. [2]

[2] Odkládání webhooks do fronty nakonec nebylo implementováno. Webhooks jsem ale implementoval dříve, než k rozhodnutí neimplementovat frontu webhooks došlo.



Obrázek 4.5 Vytváření Toggl Track webhook, vytvořeno pomocí [67], notace BPMN [68]

Proti tomuto řešení by se dalo namítat, že je třeba více synchronních volání vzdálené služby. To ale není nutně pravda. Webhooks vždy neobsahují všechna data potřebná pro Timer2Ticket. Například Jira webhook související s časovým záznamem neobsahuje id projektu, bez kterého ale nelze časový záznam správně přesunout do druhého nástroje. Takže by i tak bylo nutné se Jira doptávat.

Z těchto důvodů mi přišlo rozumnější provádět ověření volajícího synchronně stejně, jako je tomu při klasické synchronizaci.

4.3.7 Frontend

V uživatelském rozhraní je nově návod, jak zprovoznit webhooks. Pro Jira uživatel k dispozici URL, na které mají být webhooks posílány. V rámci URL je i zakódováno, ke kterému spojení webhook patří. Pro Toggl Track je zde tlačítko, kterým uživatel webhook vutvoří pomocí Timer2Ticket.

4.3.8 Api

Vrstva Api při obdržení nového webhook vytvoří objekt obsahující id volajícího objektu, zda se jedná o projekt, úkol, nebo časový záznam, datum a čas, ke kterému spojení a službě se webhook váže a o jaký typ události se jedná (created, updated, deleted). Tuto informaci následně předá core ke zpracování. Objekt zprávy 4.1 k nahlédnutí je v (pseudo)formátu typeScript [9] objektu.

Podmínky pro přijetí webhook

Aby byl webhook přijat k dalšímu zpracování, musí splnit následující podmínky:

Výpis kódu 4.1 Zpráva o příchozím webhook vytvořena API

```
{
    id: string | number,
    type: "issue" | "project" | "timeEntry",
    timestamp: Date,
    connectionId: ObjectId,
    serviceNumber! number,
    event: created | updated | deleted
}
```

Odpovídají členství: Uživatel, k jehož spojení se webhook váže musí mít členství Senior, jinak bude webhook zahozen.

Poslední synchronizační job skončil úspěchem: Není možné zpracovat webhooks, pokud předchozí synchronizace nebyla úspěšná. Data v databázi Timer2Ticket totiž nebudou v takovém případě odpovídat realitě a zpracováním webhooks by se tento stav pouze zhoršil.

Nesmí zrovna běžet job: Timer2Ticket je sám při průběhu jobu zdrojem pro vyvolání webhooks ve vzdálené službě. Proto nedává smysl zpracovávat webhooks, pokud zrovna běží job na daném spojení. Pokud se v této době odehraje změna ve službě, která by měla být přes webhook řešená, tak bude změna patrná až při další pravidelné synchronizaci (za hodinu).

Webhook prošel kontrolou proti cyklení: Přijetí webhook a jeho zpracování vyvolá potenciálně nový (ale se stejným obsahem) webhook ze druhé služby ve spojení. Proto bude možné konkrétní objekt pomocí webhooks aktualizovat až po nějakém čase od jeho poslední aktualizace (1 minuta).

4.3.9 Core

Vrstva Core má za úkol, aby byla data z webhook propsána do druhé služby. Poté co Core obdrží od Api informace o obdrženém webhook ve formátu 4.1. Na základě dat v ní se zeptá služby, která webhook vyvolala, na objekt kterého se tato zpráva týká (podle id). Díky tomu zároveň dojde k ověření volající služby. S obdrženým objektem dále pracuje podle pravidel synchronizace definovaných Vítem Štefanem [1].

4.3.9.1 Odložené zpracování

Ukládání webhooks do fronty a jejich následné pozdější zpracování nebylo implementováno. Kvůli vyřešení problému s množstvím API volání pro získání časových záznamů z Jira (viz 4.1.5) ztratila implementace fronty webhooks na výkonnostním významu. Navíc se ukázalo, že není možné se na webhooks stoprocentně spolehnout tak, aby bylo možné vynechat klasickou synchronizaci (viz 4.3). Z těchto důvodů si myslím, že ani do budoucna nemá smysl tuto funkcionalitu implementovat.

4.4 Aktualizace databázového schématu

Ne všechny změny zmíněné v návrhu databáze 3.8 byly nakonec implementovány. V této části popíšu stav aktualizovaného databázového modelu na základě skutečné implementace. Implementované změny souvisí s potřebou nových atributů pro nástroj Jira a pro spojení dvou projektových nástrojů. Seznam implementovaných změn je následující:

- Změna typu atributu *UserId* v *ServiceConfig* z *number* na *number* nebo *string*.

- Rozšíření *ServiceConfig* o *FallbackIssue*.
- Rozšíření *ServiceConfig* o *CustomField*.
- Rozšíření *ServiceConfig* o *IssueState*.
- Rozšíření *ServiceConfig* o seznam ignorovvných stavů ignoredIssueState.
- Rozšíření *Connection* o *projectMappings*.

4.4.1 Změny oproti návrhu

Z různých důvodů nebyly implementovány všechny změny podle návrhu 3.8. Zde uvádím jejich výčet spolu s důvody, proč k implementaci nedošlo.

WebhookQueue: Webhooks nakonec nejsou ukládány pro pozdější zpracování 4.3, proto již nedává tato kolekce smysl.

Atribut *webhookON* v Synced Service: Tento atribut měl Timer2Ticket indikovat, zda se má u dané služby počítat s webhook. Implementace 4.3 ale nakonec tento atribut nepotřebuje.

User Mappings: Mapování uživatelů nebylo v rámci této práce nakonec implementováno.

Notification Settings: Notifikace nakonec nebyly v této práci implementovány, proto ani nebyla rozšířena databáze.

Atribut *notificationHandled* v Kolekci Joblog: Notifikace nebyly v rámci této práce implementovány.

4.4.2 Nový model Databáze

Pro lepší přehlednost zde přikládám i Nový model databáze, který je oproti návrhu (sekce 3.8) očištěn o neimplementované změny. Model 4.6 tedy popisuje skutečný současný stav databáze a při dalším rozšiřování nástroje Timer2Ticket z něj lze vycházet. Atributy přidané v této práci jsou stejně jako v návrhu znázorněny modrým písmem.

Obrázek 4.6 Schéma databáze, vytvořeno pomocí [59], bez notace

4.4.3 Migrace databáze

Všechny přidané atributy do databáze jsou pouze rozšířením stávajících kolekcí o (z pohledu Timer2Ticket) nepovinná data. V implementaci navíc probíhá kontrola, zda jsou potřebná data v databázi skutečně uložena a případné chyby jsou ošetřeny. Není proto potřeba databázi migrovat a nová verze Aplikace může bez problémů fungovat na původní databázi.

4.5 Přínos implementovaných změn pro Jagu s. r. o.

Implementací synchronizace časových záznamů mezi Jira a Redmine dojde na vzorovém projektu k ušetření asi 500 minut práce (podle analýzy vytížení zdrojů Jagu s. r. o. viz 2.7). To je více než 8 hodin měsíčně na vzorovém projektu. Tento čas není v pravém slova smyslu ušetřen, ale tolik času by bylo třeba před zavedením Timer2Ticket vynaložit na práci, kterou Timer2Ticket nyní odvede.

Kopírování úkolů a synchronizace jejich atributů (mimo časové záznamy) mezi systémy nebylo implementováno, protože by to nebylo vhodnou součástí aplikace (viz 2.6.7). Pro tento účel ale může vzniknout samostatný nástroj. Pro Jagu s. r. o. tak zůstává nutností manuálně tyto úkoly vytvářet a provazovat. Timer2Ticket ale na proces neklade žádné další nároky a přináší lepší data o času stráveném na jednotlivých úkolech pro zákazníka.

Kapitola 5
Testování

Cílem testování bylo ověřit funkčnost především integrací jednotlivých služeb, proto bylo testování převážně manuální. Navíc jsem výslednou implementaci podrobil uživatelským testům, které ukázaly na nedostatky implementace z pohledu použitelnosti.

5.1 Lokální testy

Vzhledem k povaze programovaných částí jsem nevyužil automatické unit testy. Velká část implementace totiž pracuje s dalšími aplikacemi (především Jira). Pro testování jsme měl ve všech integrovaných systémech (Toggl Track, Jira, Redmine) vytvořený vždy alespoň jeden účet a pomocí nich testoval propisování dat.

Pro testování komunikace vrstvy Core s Jira jsem využil testovacího souboru, ve kterém byly postupně volány jednotlivé metody pro komunikaci s Jira. Následně jsem manuálně kontroloval, zda vytváření, aktualizace a mazání objektů probíhá tak, jak mělo. K automatizaci tohoto testování by bylo třeba vytvořit speciální testovací účet ve službě, a po každém spuštění testovacího skriptu vracet testovací projekty do původního stavu. Tento testovací soubor je (bez přihlašovacích údajů) k nalezení na přiloženém médiu. V rámci testování jsem musel upravovat viditelnost jednotlivých metod, proto není tento kód momentálně spustitelný.

Api rozhraní aplikace jsem testoval buď voláním z uživatelského rozhraní, nebo s využitím služby Postman [78]. Opět se ale nejedná o automatické testy z podobných důvodů jako u testování Core. Uživatelské rozhraní jsem nejdřív testoval sám manuálně, následně bylo podrobeno uživatelským testům (následující kapitola).

5.2 Testy webhooks

Protože je pro testování webhooks potřeba mít veřejnou IP adresu, využil jsem platformu *ngrok* [79], která přesměrovává dotazy na lokální vývojové prostředí na mém počítači.

Testování samotné nebylo automatizováno, ale využíval jsem sady dotazů v nástroji Postman [78] a manuálně kontroloval všechny dotčené služby pro všechny možnosti událostí (create update, delete) a všechny sledované objekty (projekt, úkol, časový záznam).

5.3 Uživatelské testy

Uživatelské testy jsem provedl kvůli ověření, zda aplikaci uživatelé rozumí a chápou kroky v ní. Aby testy dávaly smysl, musí být splněny následující počáteční podmínky.

- Uživatel musí být přihlášen do Timer2Ticket.
- Uživatel musí být přihlášen do Toggl Track.
- Uživatel musí být přihlášen do Jira.
- Uživatel musí být přihlášen do Redmine.

Testování se zúčastnili celkem čtyři testeři. Tři z nich už měli nějakou zkušenost se stávající verzí Timer2Ticket. S těmi jsem byl v kontaktu už při sběru požadavků na tuto práci. Pro testování jsem si je vybral proto, že jsem takto schopen určit, zda jsem splnil jejich požadavky na Timer2Ticket. Testování tak zároveň posloužilo jako zpětná vazba pro mě ohledně splnění jednotlivých požadavků z analýzy 2.5. Nevýhodou je, že aplikaci Timer2Ticket už znají, vědí k čemu slouží a čeho by měli v průběhu testů dosáhnout. Mohou tak snadněji přehlédnout některé nedostatky.

Čtvrtý tester s aplikací Timer2Ticket žádné zkušenosti nemá. Proto doufám, že mi poskytne jiný typ zpětné vazby. Určitě by bylo vhodné spolupracovat s více nezasvěcenými testery, není ale jednoduché najít uživatele, který by řešil problém, jehož řešení Timer2Ticket nabízí (viz 2.4.2) a zároveň se z nástrojem nesetkal.

5.3.1 Průběh testů

Každému z testerů jsem před začátkem testu vysvětlil účel nástroje Timer2Ticket (podle jeho znalosti). Dále jsem mu řekl, že není cílem testovat jeho, ale nástroj. A že pokud něčemu nerozumní, že to není jeho chyba. Následně jsem jim dával úkoly podle scénářů níže a zapisoval si zpětnou vazbu. Testy jsem nahrával pro pozdější analýzu, kde se který tester zarazil a co mu dělalo problémy. Nahrávky jsou k dispozici na přiloženém médiu.

Ne se všemi testery jsem prošel všechny testovací scénáře. S testováním jsem totiž začal dříve, než byly dokončené webhooks. U jednoho z testerů jsem také z časových důvodů vynechal testování spojení projektových nástrojů. Testování probíhalo na mém počítači vždy při osobním setkání. Testeři využívali v testovaných aplikacích mé účty.

5.3.2 Konfigurace spojení Jira – Toggl Track

Cílem tohoto scénáře je otestovat funkčnost a srozumitelnost formuláře pro synchronizaci služby Jira.

Odhadovaný čas

- 5 minut

Počáteční bod

- Uživatel je přihlášen do nástroje Timer2Ticket.
- Uživatel se nachází na obrazovce „Spojení“ (Connections).

Koncový bod

- Uživatel má vytvořené nové spojení mezi Jira a Toggl Track.

Instrukce pro testera

Spojte pomocí nástroje Timer2Ticket nástroje Jira a Toggl Track, aby se vám mezi nimi synchronizovaly časové záznamy.

Očekávané kroky

1. Stisknutí „ADD NEW“ tlačítka v pravém horním rohu obrazovky.
2. Vybrání nástroje „Toggl Track“ z levé nabídky nástrojů.
3. Vyplnění formuláře spojení pro Toggl Track platnými údaji.
4. Vybrání nástroje „Jira“ z pravé nabídky nástrojů.
5. Vyplnění formuláře spojení pro Jira platnými údaji.
6. Kliknutí na tlačítko „SAVE“ v dolní části konfigurace spojení.

5.3.3 Omezení množiny synchronizovaných úkolů

Cílem tohoto scénáře je otestovat srozumitelnost možnosti omezení množiny synchronizovaných stavů úkolů a následné provedení synchronizace mezi nástroji nakonfigurovaných v předchozím scénáři 5.3.2.

Odhadovaný čas

- 5 minut

Počáteční bod

- Uživatel je přihlášen do nástroje Timer2Ticket.
- Uživatel se nachází na obrazovce „Spojení“ (Connection).
- Uživatel má úspěšně vytvořené spojení ze scénáře 5.3.2.

Koncový bod

- Uživateli se dále nebudou vytvářet v Toggl Track tagy hotových úkolů z Jira.
- K úkolu vytvořenému v Jira je vytvořen odpovídající tag v Toggl Track.
- Časový záznam z Toggl Track je přenesen do Jira.

Instrukce pro testera

U dříve vytvořeného spojení nakonfigurujte, aby k hotovým úkolům z Jira nebyly vytvářeny tagy v Toggl Track. Následně vytvořte v Jira úkol (nesmí být hotový) a v Toggl Track k němu vykažte čas.

Očekávané kroky

1. Otevření záložky nastavení synchronizovaných stavů v rámci spojení.
2. Zrušení označení hotového stavu.
3. Otevření Jira a založení nového úkolu.
4. Spuštění synchronizace konfiguračních objektů v Timer2Ticket tlačítkem „SYNCHRONIZE CONFIGURATION OBJECTS“.
5. Otevření Toggl Track a vytvoření časového záznamu s příslušným tagem.

6. Uložení nového časového záznamu.
7. Spuštění synchronizace časových záznamů v Timer2Ticket tlačítkem „SYNCHRONIZE TIME ENTRIES“.
8. Otevření Jira a kontrola časových záznamů.

5.3.4 Konfigurace synchronizace Jira – Redmine

Cílem tohoto testovacího scénáře je otestovat vytváření spojení mezi projektovými nástroji. Především srozumitelnost vybrání mapovacího vlastního pole a vytváření párů projektů mezi nástroji Jira a Redmine.

Odhadovaný čas

- 10 minut

Počáteční bod

- Uživatel je přihlášen do nástroje Timer2Ticket.
- Uživatel se nachází na obrazovce „Spojení“ (Connection).

Koncový bod

- Uživatel má vytvořené nové spojení mezi Jira a Redmine.

Instrukce pro testera

Vytvořte spojení mezi Jira a Redmine. Jako mapovací vlastní pole vyberte u Redmine „Jira link“, u Jira „Redmine link“. Vyberte alespoň jeden pár projektů, které budete synchronizovat.

Očekávané kroky

1. Stisknutí „ADD NEW“ tlačítka v pravém horním rohu obrazovky.
2. Vybrání nástroje „Jira“ z levé nabídky nástrojů.
3. Vyplnění formuláře spojení pro Jira platnými údaji.
4. Vybrání nástroje „Redmine“ z pravé nabídky nástrojů.
5. Vyplnění formuláře spojení pro Redmine platnými údaji.
6. Výběr synchronizačního vlastního pole pro Jira.
7. Výběr synchronizačního vlastního pole pro Redmine.
8. Kliknutí na Projekt Jira, kliknutí na projekt Redmine a následně tlačítko „ADD PAIR“.
9. Kliknutí na tlačítko „SAVE“ v dolní části konfigurace spojení.

5.3.5 Vytvoření páru úkolů

Cílem tohoto testovacího scénáře je otestovat srozumitelnost provazování úkolů ve službách Jira a Redmine tak, aby mezi nimi mohl Timer2Ticket provádět synchronizaci časových záznamů.

Odhadovaný čas

- 5 minut

Počáteční bod

- Uživatel je přihlášen do nástroje Timer2Ticket.
- Uživatel se nachází na obrazovce „Spojení“ (Connection).
- Uživatel má úspěšně vytvořené spojení ze scénáře 5.3.4.

Koncový bod

- V nástroji Jira i Redmine existují úkoly, které jsou v Timer2Ticket provázány.
- Mezi úkoly proběhla synchronizace časových záznamů.

Instrukce pro testera

Pro dříve vytvořené spojení mezi Redmine a Jira vytvořte pár úkolů (pomocí odkazu na druhý úkol) pro vybrané projekty. Vytvořte pro to nové úkoly v obou systémech.

Očekávané kroky

1. Otevření projektu v Jira a založení nového úkolu.
2. Zkopírování odkazu na tento úkol.
3. Otevření projektu v Redmine a založení nového úkolu.
4. Vložení odkazu na úkol v Jira na příslušné mapovací vlastní pole.

   Alternativně lze odkaz na Redmine úkol vložit do Jira.

5.3.6 Nastavení webhooks

Cílem tohoto testovacího scénáře je otestovat nastavování webhooks v nástroji Timer2Ticket.

Odhadovaný čas

- 5 minut

Počáteční bod

- Uživatel je přihlášen.
- Uživatel se nachází na obrazovce „Spojení“ (Connection).
- Uživatel má úspěšně vytvořené spojení ze scénáře 5.3.2.

Koncový bod

- Uživatel má nakonfigurovaný webhook pro Toggl Track.
- Uživatel má nakonfigurovaný webhook pro Jira.

Instrukce pro testera

Zprovozněte v nástroji Timer2Ticket pro spojení z prvního scénáře webhook pro Jira i pro Toggl Track.

Očekávané kroky

1. Otevření nastavení rozvrhu synchronizace spojení „Synchonization Schedule“.
2. Kliknutí na „CREATE WEBHOOK“ v sekci Toggl Track.
3. Přečtení si instrukcí pro vytvoření webhook v Jira.
4. Zkopírování url pro webhook.
5. vyhledání nastavení webhooks v Jira.
6. vytvoření webhook se správným URL a událostmi.

5.3.7 Výsledky testování

Jiří Hunka

S vedoucím práce jsem testoval 27. února a byl v pořadí mým prvním testerem. Testoval všechny scénáře, kromě nastavení webhooks, protože ty v danou chvíli nebyly ve webovém rozhraní implementovány. Kromě testování uživatelské přívětivosti jsem si ještě z tohoto testu odnesl připomínky k tomu, jak testování provádět a testy jsem na základě toho upravil.

Jiří Hunka je businessovým vlastníkem aplikace a jeho požadavky jsem zjišťoval v analýze (viz 2.5). Nebyl ale u finálního ladění těchto požadavků. Ví tedy, k čemu je aplikace dobrá a jaké jsou její přínosy. Nezná ale implementační detaily a úvahy za nimi.

Při testu synchronizace Toggl Track a Jira, měl problém najít v obou službách API klíče. Stěžoval si na špatné nápovědy, kde klíče hledat. Protože ale již používá synchronizaci mezi Toggl Track a Redmine, tak neměl žádný problém s přenesením časového záznamu ve druhém testu. Problém ale byl při úkolu odebrat některé stavy ze synchronizované množiny. Nebylo mu totiž jasné, co se danou operací stane. Bude to zde třeba lépe vysvětlit v nápovědě.

Při nastavování synchronizace Jira a Redmine už neměl problém s nastavením přihlašovacích údajů do služeb. Následně ale nevěděl, k čemu je vlastní pole, které vybírá. V Redmine automaticky vybral pole „Jira link“, ale u Jira zvolil „Time Entries“, protože myslel, že do tohoto pole bude čas přenášen. Párování projektů proběhlo bez problémů a následné spojení úkolů přes odkaz už také (odkaz na Jira úkol do Redmine).

Při konfiguraci obou spojení uvažoval Jiří Hunka nad tím, který nástroj má být „první“ a který „druhý“. Přitom na tom vůbec nezáleží, ale možná by bylo vhodné to lépe s uživatelem komunikovat. Dostal jsem také odezvu na průběh testu, a co mám do příště zlepšit. Měl bych testerovi především lépe vysvětlit, v jaké je situaci a čeho chce dosáhnout.

Na základě tohoto prvního testu jsem přidal do aplikace Timer2Ticket stránku s nápovědou, je tam zjednodušeně popsáno, jak probíhá synchronizace a která data jsou od uživatele potřeba. Navíc jsem upravil některé nápovědy tak, aby byly uživateli více nápomocné. Takto upravenou aplikaci jsem testoval s dalšími uživateli. Zbylé nedostatky shrnu až po dalších testech. Špatnou nápovědu ale považuji za zásadní, proto byla implementovaná okamžitě.

Sára Sovičková

V pořadí druhým testerem byla Sára Sovičková, která mi v analýze 2 výrazně pomohla při sběru požadavků. Zatím byla zvyklá používat pouze starou verzi Timer2Ticket od Víta Štefana [1] a bylo pro ni překvapením nové webové rozhraní.

První test (vytvoření spojení mezi Jira a Toggl Track) splnila bez větších problémů. Úprava nápověd, kde hledat klíče po prvním testu byla určitě k lepšímu. Jediné, kde odhalila problém, bylo založení nového spojení, které založila dvakrát. Tlačítko pro přidání se nachází v pravém horním rohu obrazovky a nové spojení je založeno na konci seznamu již stávajících spojení (ve spodní části obrazovky).

Odstranění hotových úkolů ze synchronizace ve druhém testu proběhlo bez problémů, ale nebyla si jistá, co tato změna pro synchronizaci přinese. Navíc po odebrání stavu ze synchronizace klikala na uložení spojení, což je v tu dobu zbytečné.

Při zprovozňování synchronizace mezi Jira a Redmine došlo k chybnému vyplnění některého z políček ve formuláři Jira. Timer2Ticket bohužel nemá jak zjistit, které z těchto polí je vyplněno chybně a tak to ani nelze říct uživateli. Při výběru mapovacího vlastního pole nevěděla, k čemu to je dobré a jak to bude použité. Poté chvíli nevěděla, co se od ní očekává při párování projektů a následně si stěžovala na zavření boxu spojení ihned po jeho uložení.

U párování úkolů bylo zajímavé, že automaticky a ze zvyku vložila odkaz na vedlejší úkol do obou systémů (Jira odkaz do Redmine a Redmine odkaz do Jira). Timer2Ticket stačí pouze jeden z nich, ale pokud jsou úkoly provázány do kříže, tak to synchronizaci nevadí. Líbila by se jí možnost v nastavení, že by Timer2Ticket druhý odkaz doplnil sám.

Na rozdíl od prvního testera byly již k testování připraveny i webhooks. Nestihl jsem ale implementovat odezvu na zkopírování linku pro Jira webhook URL, což se samozřejmě ukázalo jako chyba. Problém byl i s návodem na založení webhook, hledání jejich nastavení v Jira bylo delší, než jsem předpokládal. Bude třeba opět vylepšit nápovědu. Při konfiguraci webhook byl také problém vybrat všechny potřebné události, na které má Jira webhook poslat. Sára Sovičková by ocenila tuto informaci ve webovém rozhraní výraznější (nachází se pod odkazem, který zkopírovala a z Timer2Ticket odešla). Kvůli tomu nenakonfigurovala webhook správně, což ale potvrzuje mou obavu, že se na webhooks nedá spolehnout, pokud nad nimi nemá Timer2Ticket plnou kontrolu.

Celkově byly hlavním problémem opět nápovědy a vysvětlivky, k čemu je které pole dobré a co se od uživatele očekává. To jsem se sice snažil vylepšit po prvním testu, zároveň už byla hotová stránka s nápovědou, ale to Sáru Sovičkovou při testu ani nenapadalo hledat. Ocenila by také, kdyby Timer2Ticket po vytvoření páru úkolů při synchronizaci mezi projektovými nástroji uměl vložit odkaz i do druhé služby. To vychází z toho, jakým způsobem nástroje na projektech provazuje. Zákazníkovi do Jira vloží odkaz do Redmine, aby mohl kontrolovat čas a vývojářům vloží do Redmine odkaz na Jira, aby mohli sledovat stav úkolu. Toto by ale podle mě nemělo být úkolem nástroje Timer2Ticket.

Oldřich Malec

Dalším testerem byl Oldřich Malec, který je produktovým vlastníkem aplikace Timer2Ticket a požadavky z analýzy 2 jsou z velké části jeho požadavky.

V testu nastavení spojení mezi Toggl Track a Jira se nejdříve pokoušel najít Toggl Track API klíč podle paměti, po neúspěchu si přečetl nápovědu a klíč našel bez problémů. Podle nápovědy našel správně i Jira API klíč, ale ocenil by, kdyby dostal rovnou odkaz a nemusel klíče hledat. Chtěl si nastavit i fallback úkol, to se podařilo správně, ale nebyl si jistý, jestli lze úkol později měnit. Bylo by vhodné sem doplnit nápovědu. Jinak proběhlo nastavení spojení bez problémů.

Odebrání hotových úkolů ze synchronizace objektů proběhlo bez problémů. Ocenil by ale výraznější odlišení vybraných a nevybraných stavů úkolů, například přidáním checkboxu. Není také jasné, jak se bude nástroj chovat při mazání a má z toho obavy.

Při vytváření nového spojení pro synchronizaci Jira a Redmine bylo toto spojení založeno dole, pod stávajícími spojeními. Mělo by to podle něj být nahoře, aby bylo lépe vidět. Z mně neznámého důvodu se nenačetly správně vlastní pole ani projekty ani z Redmine, ani z Jira. Pro takovýto případ je tam tlačítko na refresh, to ale Oldřich Malec nevyužil a musel jsem mu to poradit. Párovací vlastní pole zvládl vybrat bez problémů a věděl, k čemu a proč je vybírá.

Nevěděl ale, že výběr vlastního pole následně omezí výběr projektů pouze na ty, které toto vlastní pole mají. Problém také nastal při párování projektů, kdy neočekával, že bude nutné potvrdit pár tlačítkem uprostřed.

Párování úkolů ve vybraných projektech proběhlo bez problémů, podobně jako Sára Sovičková ale vyplňoval odkazy na úkoly do obou služeb. Shodli jsem se ale, že by přenášení odkazu z jedné služby do druhé by nemělo být úkolem nástroje Timer2Ticket. Ten se má starat o časové záznamy.

Při nastavování webhooks mu nestačila informace o zkopírování zobrazenou fajfkou u odkazu, ale ocenil by notifikační snackbar nahoře, jako u ostatních notifikací. Ocenil by také odkaz na nastavení webhooks a text by mohl být schovaný pod nápovědním íčkem. Také by se mu líbilo automatické vyplnění formuláře konfigurace webhook při kliknutí na odkaz (který tam zatím není). Také si myslí, že by se dala správnost konfigurace webhook otestovat několika voláními. Podle mě to není rozumné, protože by to znamenalo vytváření a mazání objektů v cizím projektovém nástroji.

Celkově navrhl po testu několik drobných úprav, především přidání klikacích odkazů na místa, kde uživatel musí něco dělat nebo hledat (API klíče, webhooks). Nelíbí se mu vytváření nových spojení dole a některé informační notifikace. Navíc si není vždy jistý, co která operace a nastavení znamená a jaké bude chování.

Adam Šercl

Adam je můj kamarád, který se po absolvování medicíny rozhodl pracovat jako frontendový vývojář. Na testování jsem si ho vybral, protože na rozdíl od předchozích testerů aplikaci Timer2Ticket nikdy nepoužíval, ale zároveň používá nástroj pro měření času podobný Toggl Track (Clockify [80])a v práci projektový nástroj Jira.

Timer2Ticket jsme testovali 26. března. Začal jsem detailním popisem, k čemu nástroj Timer2Ticket je a co od něj budu při testování požadovat. Vysvětlil jsem mu, k čemu jsou jednotlivé nástroje, se kterými bude pracovat a nechal ho před začátkem testu projít aplikaci Timer2Ticket. Kvůli časové tísni jsme stihli projít pouze scénáře 1, 2 a 5. Vynechali jsme tak synchronizaci projektových nástrojů. Ta pro něj navíc byla při vysvětlování smyslu nástroje těžko pochopitelná.

Při prvním testu nejdříve správně vytvořil spojení a vybral nástroje k synchronizaci. Následně ale, jako všichni testeři, ignoroval nápovědy a vydal se hledat API klíč do aplikace. V Toggl Track zvládl relativně rychle najít API klíč i bez nápovědy, ale z Jira se časem vrátil do Timer2Ticket, přečetl si nápovědu a podle ní už správně našel stránku s API klíči. Odkazu na stránku s API klíčem si v Timer2Ticket nevšiml. Následně správně pochopil, k čemu je fallback úkol u Jira, správně ho použil a spojení uložil.

Následně šel do Toggl Track hledat úkoly mezi tagy. Ty tam ale nebyly, tak spustil synchronizaci časových záznamů a po druhé kontrole i synchronizaci konfiguračních objektů, která vytvořila tagy v Toggl Track. Stěžoval si ale na to, že se to nestalo okamžitě.

Druhým úkolem bylp odebrat hotové úkoly ze synchronizace. Nejdřív bylo kvůli chybě potřeba se odhlásit a znovu přihlásit do Toggl Track, protože se nenačítal workspace. Po opravení chyby Adam Šercl nejdříve zkoumal všechny atributy spojení a pak s jistotou zrušil označení „Done“ úkolů. Byl ale překvapený, že tagy nezmizely z Toggl Track okamžitě a ocenil by tuto informaci v nápovědě. Následně se mu podařilo naměřit čas v Toggl Track k některému z úkolů a po chvíli čekání a hledání v Jira mu došlo, že musí manuálně spustit synchronizaci.

Následně jsme přeskočili k testu konfigurace webhooks. Konfigurace Toggl Track proběhla správně stisknutím tlačítka pro jeho vytvoření. U Jira byl problém s hledáním nastavení webhooks. Po nalezení ale nakonfiguroval webhook správně.

Celkově Adam zvládl všemi třemi připravenými testy projít, i když místy s obtížemi. Přes veškerou mou snahu vylepšovat nápovědy v tooltipech se stále ukazují být nedostatečné. Adam Šercl zde navrhl možnost zobrazení nápovědy v modálním okně. Ty pojmou více textu a bude do nich možno jednodušeji vložit odkaz, kterého si uživatel všimne.

Souhrnné výsledky testování

Z testování jsem si odnesl tyto zásadní poznatky:

Pořadí nástrojů: Synchronizace probíhá, bez ohledu na pořadí nástrojů, vždy stejně. To ale není uživateli sděleno.

Co bude synchronizace znamenat: Nezasvěcený uživatel nemá jak zjistit, co bude synchronizace obnášet.

Nedostatečné nápovědy: Nápovědy by měli uživatele lépe nasměrovat, kde například hledat API klíč, nebo k čemu je mapovací vlastní pole u synchronizace projektových nástrojů.

Sbalení boxu spojení: Ihned po konfiguraci dojde k zavření boxu spojení, což schová možnosti dalšího nastavení (harmonogram, stavy, ...). Navíc při vytvoření nového spojení je nový box vytvořen dole. V případě, že má uživatel již více spojení, tak vůbec nemusí být vidět.

Synchronizace stavů úkolů: Bude třeba výrazně lépe vysvětlit, co se reguluje výběrem stavů, které chce uživatel synchronizovat. Z nápovědy nebylo testerům jasné, že se jedná o související tag v Toggl Track.

Výběr synchronizačního vlastního pole: Uživateli nemusí být jasné, k čemu synchronizační vlastní pole je a proč ho má vybírat.

Ukládání v rámci spojení má 2 různé logiky: Spojení ukládá uživatel stisknutím tlačítka „Save“. Nastavení harmonogramu synchronizace a výběr stavů se ale ukládá automaticky při změně a uživatel je o tom informován (podle testů zjevně snadno přehlédnutelnou) hláškou v pravém horním rohu.

Pojmenování spojení: Spojení jsou nyní pojmenována podle toho, které domény (pracovní prostředí) spojuje. To podle testů uživatele ale nezajímá o ocenili by lepší pojmenování spojení.

Automatické vyplňování selectů: Při konfiguraci Redmine i Toggl Track dochází k výběru workspace z nabídky. Pokud je pouze jedna možnost, tak by tato možnost měla být vybraná automaticky.

Úpravy na základě testů

Na základě výsledků testů jsem implementoval některé úpravy. Nejedná se ani z daleka o opravu všech chyb, které testy odhalily, protože by implementace byla příliš časově náročná. Možnosti, jak opravit stávající chyby diskutuji v části 6.2. Úpravy jsem provedl u těchto zásadních bodů:

Úprava nápověd: Problematické stávající nápovědy byly upraveny, aby byly lépe popisné. Navíc bylo vytvořeno několik dalších nápověd v místech, kde jsem původně nepředpokládal, že budou potřeba. Například vysvětlení párování projektů.

Přidání odkazů na API klíče: Tam kde to bylo možné (odkaz je stálý) byly do tooltipů přidány odkazy na získání API klíče. Například v Jira byl problém s jeho hledáním, ale odkaz na nastavení, kde API klíč vygenerovat je pro všechny uživatele stejný. Nově tedy může uživatel při konfiguraci pouze kliknout na odkaz.

Přidání stránky „Help“: Kromě úprav nápověd jsem navíc přidal stránku, na které je (detailněji než v tooltipech) vysvětleno chování aplikace. Stránka je dostupná v levém navigačním menu.

Kapitola 6

Další rozvoj

V této kapitole se věnuji dalšímu rozvoji nástroje Timer2Ticket. Je zde návod, jak do aplikace přidat další nástroje k synchronizaci a také jakým směrem by se měl ubírat další vývoj. To jsem navrhl na základě sesbíraných požadavků, které jsem nestihl implementovat a na základě výsledků uživatelských testů.

6.1 Rozšíření Timer2Ticket o další nástroj

Pro přidání nového nástroje je třeba udělat stejné kroky, které popisují Vít Štefan [1] a Martin Paul [5]. Navíc ale přibyly možnosti spojit dva projektové nástroje, implementace webhooks a filtrace úkolů k synchronizaci podle vybraných stavů.

6.1.1 Přidání projektového nástroje

Pro správné přidání projektového nástroje je navíc (oproti přidání nástroje pro měření času) potřeba ve vrstvě Api doplnit endpoint pro informování Frontend o vlastních polích projektu. Podobně, jako to dělají endpointy *GET jira_projects* a *GET redmine_projects* v souboru *synced_services_config*. Dále je třeba upravit funkce v souboru *Ticket2Ticket_service* tak, aby počítala s další službou.

6.1.2 Přidání webhooks

Webhooks nejsou nutnou součástí přidání dalšího nástroje, pokud ale nástroj webhooks podporuje, tak jejich využití zvedne komfort uživatelů.

Frontend

Je třeba informovat uživatele, jak webhook nastavit. To je nutné přidat do stránky *Help*, a také přímo do *WebhookSettings* v rámci nastavení rozvrhu synchronizace.

Api

V API je třeba přidat endpoint pro konkrétní službu a při volání webhook vytvořit objekt pro předání Core. Ten musí zkontrolvat členství, zvalidovat webhook a informovat Core.

Core

V Core bude možná nutné upravit funkci *handleWebhook* ve třídě *WebhookHandler* tak, aby dokázala zpracovat i volání jiných služeb.

6.1.3 Filtrace úkolů podle stavu

Filtrací může uživatel omezit úkoly z projektových nástrojů, proto nemá smysl filtraci implementovat (za stávajících pravidel synchronizace) pro nástroje na měření času.

Frontend

Je třeba upravit komponentu *IssueStateForToolSelector* tak, aby uměla vykreslit možné stavy úkolů i jiného nástroje, než je Jira.

Api

V rámci Api je nutné zajistit, aby frontend dostal informaci o stavech projektů podobně, jako je vyřešeno endpointem *GET jira_issue_statuses* v souboru *syneced_services_config*.

6.2 Úpravy frontend

Na základě výsledků uživatelského testování bych doporučil přepracovat webové rozhraní aplikace, nebo alespoň konfiguraci spojení. Z testů (kapitola 5.3) totiž vyšlo, že uživatelé si nejsou jisti, jak nástroj používat a ocenili by lepší vysvětlení, proč dělají některé krok, jak je mají provést a co bude následně výsledkem operace.

6.2.1 Změna konfigurace spojení

Konfiguraci spojení bych místo několika malých formulářů na jedné stránce doporučil implementovat formou několikakrokového wizard. Ten umožní uživateli lépe v každém kroku nastavování spojení vysvětlit, co se od něj požaduje a kde má tuto informaci najít. Odstraní se tak problém s příliš dlouhými nápovědami v tooltipech, které stejně uživatelé používají až ve chvíli, kdy se jim něco dlouho nedaří.

Wizard by podle mě měl mít zvlášť krok pro výběr, které služby chce uživatel synchronizovat, nastavení jedné služby, druhé služby, výběr mapovacího vlastního pole, párování projektů a případně i nastavení rozvrhu synchronizace, nebo omezení množiny stavů úkolů, kterých se synchronizace týká, případně dalších detailů synchronizace. Uživatel by tak mě získat lepší povědomí o tom, co se bude následně dít.

Myslím si, že dokud bylo součástí nástroje Timer2Ticket pouze spojení mezi Toggl Track a Redmine, nebyl Wizard potřeba, přidáním Jira se výrazně zvětšuje velikost formuláře především kvůli výběru mapovacího vlastního pole a výběru párů projektů. Přidáním možnosti synchronizovat časové záznamy mezi projektovými nástroji už není (podle výsledků testů 5.3) možné nechat uživatele pracovat s formulářem bez toho, aniž by k tomu dostal detailnější instrukce, na jejichž množství ale nebyl formulář připravený a nevešly by se tam.

6.2.2 Další změny pro spojení

Uživatel by měl mít možnost si spojení podle sebe pojmenovat. Nyní je jméno spojení vytvářeno podle domén, které synchronizuje a to může být pro uživatele nejasné.

Také by bylo dobré lépe zpřístupnit tlačítka pro spuštění manuální synchronizace. Tu by sice měl uživatel používat co nejméně, ale pokud ji hledá, tak musí nyní celé spojení rozbalit a tato tlačítka pak najde v dolní části spojení. Myslím že by tuto možnost měl mít i u spojení v nerozbaleném stavu.

6.2.3 Použití TypeScript

Doporučil bych zvážit využití TypeScript nejen pro backend (Core a API), ale i pro Frontend. To by umožnilo využívat napříč celou aplikací stejné datové typy a výrazně by to frontend zpřehlednilo a zjednodušilo další rozšiřitelnost a udržitelnost. Datové typy by například mohly být implementovány v samostatné knihovně, kterou by následně mohly využívat všechny vrstvy. Kvalita kódu na frontend navíc není příliš dobrá, nabízí se tak možnost implementovat frontend celý znovu.

6.3 Implementace dalších požadavků

V rámci analýzy jsem sesbíral více požadavků, než kolik jsem stihl implementovat. V dalším rozvoji se tedy nabízí pokračovat v těchto požadavcích. Jejich seznam včetně míry splnění je k dispozici v příloze D. Nerealizovaným požadavkům jsem vytvořil úkoly v projektovém nástroji Redmine [1] společnosti Jagu s. r. o. Zde navíc uvádím další detaily k implementaci, které mohou být užitečné pro pokračování v rozvoji nástroje Timer2Ticket.

6.3.1 Mapování uživatelů mezi projektovými nástroji

V implementaci spojení projektových nástrojů Timer2Ticket neřeší, který uživatel je autorem časového záznamu. Tuto problematiku jsem rozebíral v 3.2.3 a rozhodl jsem se mapování uživatelů vůbec neimplementovat. Požadavek 2.6.7 je určitě validní, ale tak jak mi bylo popsáno fungování v rozhovorech v analýze 2.5.1, tak si nemyslím, že za stávajících podmínek dává smysl implementovat. Myslím si, že získávání API klíčů od kolegů proto, aby mohl Timer2Ticket správně provádět synchronizaci je přinejmenším uživatelsky složité a nepřehledné. Zejména kvůli bezpečnostním otázkám. Přidaná hodnota (v obou systémech bude shodný čas i uživatelé) v popsaných podmínkách podle mě není příliš velká. Implementace mapování uživatelů tak podle mě nemá velkou prioritu pro bezprostřední rozvoj.

6.3.2 Notifikace

Notifikacím jsem věnoval sekci v analýze 2.5 i návrhu 3.4. K implementaci jsem se již nedostal. Myslím si, že notifikace by nástroji Timer2Ticket výrazně prospěly a při dalším rozvoji bych doporučil implementovat notifikace minimálně pro případy, které jsem popsal v návrhu. Je ale možné, že se objeví nové případy, které bude třeba pokrýt.

Dále bych doporučil zamyslet se nad notifikacemi hlouběji, především v kontextu úprav Frontend. Myslím, že by bylo vhodné implementovat „zvoneček“ do webového rozhraní, aby uživatel, který přijde do Timer2Ticket na základě nějaké notifikace, měl tuto notifikaci přístupnou i přímo ve webové aplikaci.

[1] Dostupné po přihlášení na adrese https://projects.jagu.cz/projects/timer2ticket

Kapitola 7
Závěr

V rámci této diplomové práce jsem nejdříve zanalyzoval a seřadil pro další implementaci projektové nástroje vhodné k přidání do Timer2Ticket. Jako nejvhodnější v analýze vyšel nástroj Jira, který jsem lépe prozkoumal a spolu s ním i existující spojení s nástrojem Toggl Track. Na základě zkoumání tohoto spojení, předchozích prací na téma Timer2Ticket a rozhovory s lidmi, kteří nástroj Timer2Ticket vlastní a používají sepsal požadavky na jeho rozšíření. Tyto požadavky jsem rozdělil do skupin podle tématu a přiřadil jim priority, ty byly následně stakeholdery odsouhlaseny.

Následně jsem navrhl, jakým způsobem upravit nástroj Timer2Ticket tak, aby vyhověl jednotlivým požadavkům. V návrhu jsem se soustředil na využití stávajících datových struktur a celkově na to, aby byl zásah do fungující aplikace co nejmenší. Navrhované změny se dotkly všech vrstev aplikace, i databáze.

V rámci implementace jsem zvládl pokrýt většinu požadavků z analýzy. Přidal jsem do Timer2Ticket nástroj Jira, včetně některých funkcionalit navíc, které původní Timer2Ticket pro spojení nepodporoval (omezení množiny stavů úkolů a mazání starých úkolů). S přidáním Jira bylo nutné implementovat i spojení dvou projektových nástrojů, které uživatelům umožní synchronizovat časové záznamy pro projektové nástroje pro projekty, které zvolí a pro úkoly, které manuálně provážou. Také jsem implementoval zpracovávání webhooks pro Toggl Track a Jira. To přineslo Timer2Ticket schopnost reagovat na změny v těchto aplikacích okamžitě, bez čekání na pravidelný job. Uživateli to přinese větší komfort.

Nevyšel mi ale čas na implementaci uživatelských notifikací. Zanechávám alespoň návrh, jakým způsobem notifikace do Timer2Ticket přidat. Dál nedošlo k implementaci „Enterprise“ verze, kterou jsme ale spolu s produktovými vlastníky zamítli již v analýze a shodli se, že by tato funkcionalita neměla být součástí nástroje Timer2Ticket. Zasloužila by si vlastní aplikaci.

Následně jsem nad implementovanými změnami provedl uživatelské testy. Zkoumal jsem především konfiguraci spojení Jira (pro spojení s Toggl Track i Redmine). Došel jsem k závěru, že by bylo vhodné přepracovat uživatelské rozhraní Timer2Ticket, protože nebylo dostatečně srozumitelné pro uživatele. Dříve zvolený model, že uživatel konfiguruje nástroje ve spojení „Vedle sebe“ bych doporučil nahradit vícekrokovým wizard. Vznikne tak prostor pro lepší komunikaci s uživatelem a vysvětlení kroků konfigurace spojení. Formuloval jsem i další doporučení pro další rozvoj nástroje Timer2Ticket.

V práci jsem také zkoumal aktuální proces synchronizace úkolů mezi Jira a Redmine ve společnosti Jagu s. r. o. Ti mají zákazníka, který pracuje a úkoly zadává v Jira, vývojový tým Jagu s. r. o. ale používá Redmine. Výsledkem je nutnost kopírovat úkoly mezi systémy. To stojí práci a čas. Implementací rozšíření Timer2Ticket o nástroj Jira a o spojení mezi projektovými nástroji tento čas sice neklesne, ale zákazník bude mít nyní přehled ve své Jira o čase stráveném nad jednotlivými úkoly. To zákazníkovi dá lepší představu o stavu projektu.

Všechny části zadání diplomové práce považuji za splněné a jsem přesvědčený, že práce přispěla k vylepšení nástroje Timer2Ticket.

Příloha A

Struktura rozhovoru k získání uživatelských požadavků

Při vytváření osnovy jsem vycházel z [49].

1. Co je cílem projektu?
2. Kdo je zákazníkem?
3. Jaký je problém se současným řešením?
4. Co by mělo řešení dělat?
   - Požadavky na synchronizaci mezi Toggl Track a Jira
   - Požadavky na synchronizace mezi projektovými nástroji
   - Nové požadavky, které nezazněly v předchozích pracích
5. Validovat pochopení požadavků
6. Navrhnout způsob pokračování a časový rámec

Příloha B

Čas strávený vytvářením úkolů mezi Jira a Redmine

Od Sáry Sovičkové jsem dostal následující data ohledně času stráveného na vytváření úkolů v Redmine podle zákazníkových úkolů z Jira.

- Počet nových úkolů měsíčně: 50
- Počet časových záznamů na úkol: 10
- Čas nutný pro zkopírování časového záznamu: 1 minuta

10/23	11/23	12/23	01/24	02/24	AVG
1.8	3.4	1.3	1.2	2.9	2.1

Tabulka B.1 Strávený čas (v hodinách) vytvářený kopírováním úkolů mezi Jira a Redmine, zdroj Sára Sovičková

Čas nutný měsíčně k přenesení časových záznamů z Redmine do Jira

$$50 \text{ úkolů} * 10 \text{ časových záznamů} * 1 \text{ minuta} = 500 \text{ minut} \approx 8.3 \text{ hodin}$$

Příloha C

Dotazník průzkumu mezi aktuálními uživateli spojení Toggl Track a Jira

Dotazník byl vytvořen pomocí Google Forms [81] a byl vložen ve formě pdf.

Integration between Toggl Track an Jira

Hello,
I am a student working on integration of Toggl Track and Jira in my master thesis. I would like to know how happy you are with current available integrations of these two tools. Results of this survey are anonymous and will be used in my thesis. Completing the form should not take more than a few minutes. Thank you for participating.

* Indicates required question

1. Do you have experience with Toggl Track and Jira integration? *

Mark only one oval.

- ◯ Yes, the one by Toggle
- ◯ No, none
- ◯ Other: _____

2. How strongly do you agree with following statements? *

Mark only one oval per row.

	Strongly disagree	Somewhat disagree	Somewhat agree	Strongly agree	NA
I am happy with this product					
It was easy to set up					
It is easy to use					
Synchronisation of tasks to Toggle works well					
It saves me a lot of time					
It is a good value for money					
I would appreciate time entries sync					

3. What other functionality would you appreciate?

4. What do you like about this integration?

5. What do you dislike about this integration?

Příloha D

Výčet požadavků sesbíraných v této práci

Zde uvádím výčet všech požadavků, které jsem v práci sesbíral. Uvádím zde jejich priority a zda byly implementovány.

id požadavku	krátký popis	priorita	implementace
TTJ01	Přidání Jira	Must have	100%
TTJ02	Konfigurace spojení	Must have	100%
TTJ03	Sync úkolů a projektů	Must have	100%
TTJ04	Sync časových záznamů	Must have	100%
TTJ05	Sync času k projektu	Must have	100%
TTJ06	Výběr Fallback issue	Should have	100% Využito jméno úkolu
TTJ07	Filtrace stavů úkolů	Could have	70% Pouze Jira
TTJ08	Hlášení chyb	Should have	0% Součást Notifikací
SPN01	Zachování T2T principu	Must have	0%
SPN01B	Vytváření objektů	Must have	100%
SPN02	Spojení Redmine a Jira	Must have	100%
SPN03	Přizpůsobení FE	Must have	100%
SPN04	Mapování 1:N	Must have	80% viz 4.2.2
SPN05	Sync za jiného uživatele	Must have	100%
SPN06	Mapování projektů a úkolů	Must have	100%
SPN07	Hlášení chyb	Should have	0% Součást Notifikací
SPN08	Čas k projektu	Should have	0% viz 4.2.1
SPN09	Mapování uživatelů	Should have	0%
NOT01	Emailové notifikace	Could have	0% Součást Notifikací
NOT02	Sentry.io notifikace	Should have	0% Součást Notifikací
PWH01	Konfigurace webhooks	Could have	100%
PWH02	Členství kvůli webhooks	Could have	100%
PWH03	Toggl Track webhooks	Could have	100%
PWH04	Jira webhooks	Could have	100%
ENT01	Nastavení jedním uživatelem	Won't have	0%
ENT02	Vytvoření úkolu	Won't have	0%
ENT03	Synchronizace atributů	Won't have	0%
ENT04	Sync časových záznamů	Won't have	0%
ENT05	Mapování uživatelů	Won't have	0%

Tabulka D.1 Seznam požadavků na Timer2Ticket

Bibliografie

1. ŠTEFAN, Vít. *Synchronizační middleware mezi projektovým systémem a aplikací na sledování času stráveného na projektech*. 2021. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
2. TOGGLE TRACK. *Toggl* [online]. [cit. 2023-10-05]. Dostupné z: `https://toggl.com`.
3. LANG, Jean-Philippe. *Redmine* [online]. [cit. 2023-10-06]. Dostupné z: `https://www.redmine.org`.
4. ČERMÁK, Jakub. *Nový frontend synchronizačního middleware Timer2Ticket*. 2022. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
5. PAUL, Martin. *Timer2Ticket II*. 2023. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
6. STRIPE, INC. *Stripe* [online]. [cit. 2023-11-02]. Dostupné z: `https://stripe.com/en-cz`.
7. FAKTUROID S.R.O. *Fakturoid* [online]. [cit. 2023-11-02]. Dostupné z: `https://www.fakturoid.cz`.
8. OKTA, INC. *https://auth0.com* [online]. [cit. 2023-10-30]. Dostupné z: `https://jwt.io`.
9. MICROSOFT. *TypeScript* [online]. [cit. 2023-10-25]. Dostupné z: `https://www.typescriptlang.org`.
10. GOOGLE. *Angular* [online]. [cit. 2023-10-25]. Dostupné z: `https://angular.io`.
11. MOZILLA FONDATION. *JavaScript* [online]. [cit. 2023-11-13]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/JavaScript`.
12. YOU, Evan. *Vue.js* [online]. [cit. 2023-11-13]. Dostupné z: `https://vuejs.org`.
13. FIELDING, Roy. *Architectural styles and the design of network-based software architectures* [online]. [cit. 2024-04-09]. Dostupné z: `https://www.epai-ict.ch/nexus/repository/course-material/m133/fielding_dissertation.pdf`.
14. OPENJS FOUNDATION. *Node js* [online]. [cit. 2023-10-25]. Dostupné z: `https://nodejs.org/en`.
15. OPENJS FOUNDATION. *Express* [online]. [cit. 2023-10-25]. Dostupné z: `https://expressjs.com`.
16. VISIONMEDIA. *Super Agent* [online]. [cit. 2024-04-09]. Dostupné z: `https://www.npmjs.com/package/superagent`.
17. TYPESTACK. *Class Validator* [online]. [cit. 2023-10-25]. Dostupné z: `https://github.com/typestack/class-validator/`.

18. CAMPBELL, Nick. *node-cron* [online]. [cit. 2023-10-25]. Dostupné z: https://github.com/kelektiv/node-cron.
19. MONGODB. *MongoDB* [online]. [cit. 2023-10-25]. Dostupné z: https://www.mongodb.com.
20. OKTA, INC. *JSON Web Token* [online]. [cit. 2023-10-30]. Dostupné z: https://jwt.io.
21. HUNKA, Jiří; MALEC, Oldřich. *Jagu s.r.o.* [online]. [cit. 2023-10-10]. Dostupné z: https://www.jagu.cz.
22. ATLASSIAN. *Jira* [online]. [cit. 2023-10-05]. Dostupné z: https://www.atlassian.com/software/jira.
23. ENLYFT. *Software Configuration Management products* [online]. [cit. 2023-10-19]. Dostupné z: https://enlyft.com/tech/software-configuration-management.
24. ENLYFT. *Project Management products* [online]. [cit. 2023-10-19]. Dostupné z: https://enlyft.com/tech/project-management.
25. SMITH, Amy. *Best Jira Alternatives Of 2023* [online]. [cit. 2023-10-19]. Dostupné z: https://www.forbes.com/advisor/business/software/best-jira-alternatives/.
26. ATLASSIAN. *Atlassian* [online]. [cit. 2023-10-19]. Dostupné z: https://www.atlassian.com.
27. RED HAT. *What is a webhook?* [online]. [cit. 2023-10-23]. Dostupné z: https://www.redhat.com/en/topics/automation/what-is-a-webhook.
28. ENLYFT. *Companies using Atlassian JIRA* [online]. [cit. 2023-10-19]. Dostupné z: https://enlyft.com/tech/products/atlassian-jira.
29. GITHUB. *GitHub* [online]. [cit. 2023-10-19]. Dostupné z: https://github.com/.
30. TIME DOCTOR. *5 best tools for GitHub time tracking* [online]. [cit. 2023-10-19]. Dostupné z: https://www.timedoctor.com/blog/github-time-tracking/.
31. ENLYFT. *Companies using GitHub* [online]. [cit. 2023-10-19]. Dostupné z: https://enlyft.com/tech/products/github.
32. GITLAB. *GitLab* [online]. [cit. 2023-10-19]. Dostupné z: https://about.gitlab.com/.
33. ENLYFT. *Companies using GitLab* [online]. [cit. 2023-10-19]. Dostupné z: https://enlyft.com/tech/products/gitlab.
34. ASANA. *https://asana.com* [online]. [cit. 2023-10-19]. Dostupné z: https://asana.com.
35. ENLYFT. *Companies using Asana* [online]. [cit. 2023-10-19]. Dostupné z: https://enlyft.com/tech/products/asana.
36. SMARTSHEET. *Smartsheet* [online]. [cit. 2023-10-19]. Dostupné z: https://www.smartsheet.com.
37. ENLYFT. *Companies using Smartsheet* [online]. [cit. 2023-10-19]. Dostupné z: https://enlyft.com/tech/products/smartsheet.
38. FREELO BAY S.R.O. *Freelo* [online]. [cit. 2023-11-16]. Dostupné z: https://www.freelo.io/cs.
39. JETBRAINS. *YouTrack* [online]. [cit. 2023-11-20]. Dostupné z: https://www.jetbrains.com/youtrack/.
40. TOGGLE TRACK. *Jira Sync* [online]. [cit. 2023-10-05]. Dostupné z: https://support.toggl.com/en/articles/4794538-jira-sync.
41. SURVIO. *How to create a product feedback survey* [online]. [cit. 2023-10-19]. Dostupné z: https://www.survio.com/en/use-cases/how-to-create-a-product-evaluation-survey.
42. REDDIT, INC. *Reddit* [online]. [cit. 2023-11-02]. Dostupné z: https://www.reddit.com.

43. REDDIT, INC. *JIRA* [online]. [cit. 2023-11-02]. Dostupné z: https://www.reddit.com/r/jira/.
44. ATLASSIAN. *ONLINE COMMUNITY PLATFORM TERMS OF USE* [online]. [cit. 2023-11-02]. Dostupné z: https://community.atlassian.com/t5/user/UserTermsOfServicePage.
45. STARŮSTKA, Jan. *Using Toggle Track integration* [online]. [cit. 2023-11-02]. Dostupné z: https://www.reddit.com/r/jira/comments/17bl7mv/using_toggle_track_integration/.
46. VICTOR, Michael Kevin. *We built a simple tool to sync Toggl and JIRA* [online]. [cit. 2023-10-11]. Dostupné z: https://blog.commutatus.com/we-built-a-simple-tool-to-sync-toggl-and-jira-9215428fa13c.
47. COMMUTATUS. *Commutatus* [online]. [cit. 2023-10-11]. Dostupné z: https://commutatus.com.
48. THORNTON, Patrick. *How to conduct user interviews* [online]. [cit. 2023-10-09]. Dostupné z: https://uxdesign.cc/how-to-conduct-user-interviews-fe4b8c34b0b7.
49. DR DEE, IFD_QUALITATIVE RESEARCH. *Semi-structured interviews guide* [online]. [cit. 2023-10-06]. Dostupné z: https://www.youtube.com/watch?v=8z8XV1S7548.
50. KRÜGER, Gerhard; LANE, Charles. *How to Write a Software Requirements Specification* [online]. [cit. 2023-10-08]. Dostupné z: https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document.
51. MINISTERSTVO SPRAVEDLNOSTI ČESKÉ REPUBLIKY. *Výpis z obchodního rejstříku* [online]. [cit. 2023-10-30]. Dostupné z: https://or.justice.cz/ias/ui/rejstrik-firma.vysledky?subjektId=354592&typ=PLATNY.
52. J. NOVÁK, I. Halaška. *Bílá kniha* [online]. [cit. 2023-10-30]. Dostupné z: https://bk.fit.cvut.cz/cz/prehled.html.
53. ČERMÁK, Jakub. Synchronizace Jira-Redmine. *Redmine*. 2023.
54. MALEC, Oldřich. Ticket 19500. *Redmine*. 2023.
55. AGILE BUSINESS CONSORTIUM. *MoSCoW Prioritization* [online]. [cit. 2023-10-30]. Dostupné z: https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html.
56. MLEJNEK, Jiří. *Analýza a sběr požadavků* [online]. [cit. 2023-10-30]. Dostupné z: https://moodle-vyuka.cvut.cz/pluginfile.php/640508/mod_resource/content/7/03.prednaska.pdf.
57. GRADY, Robert; CASWELL, Deborah. *Software Metrics: Establishing a Company-Wide Program*. Prentice Hall, 1987.
58. SENTRY INC. *Sentry* [online]. [cit. 2023-10-24]. Dostupné z: https://sentry.io/welcome/.
59. DIAGRAMS.NET. *Diagrams.net* [online]. [cit. 2023-10-24]. Dostupné z: https://app.diagrams.net/.
60. NIDA, Toggl Track. *Toggl Track Webhooks* [online]. [cit. 2023-10-25]. Dostupné z: https://support.toggl.com/en/articles/6321281-toggl-track-webhooks.
61. ATLASSIAN. *Atlassian Developer* [online]. [cit. 2023-10-06]. Dostupné z: https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/#about.
62. ATLASSIAN. *Jira API documentation* [online]. [cit. 2023-11-16]. Dostupné z: https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/#version.
63. ATLASSIAN. *Jira API v3, Authentication and authorization* [online]. [cit. 2023-11-06]. Dostupné z: https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/#about.

64. OAUTH. *OAuth* [online]. [cit. 2023-11-06]. Dostupné z: https://oauth.net.

65. TOGGL. *Toggl Track Documentation* [online]. [cit. 2023-11-26]. Dostupné z: https://developers.track.toggl.com/docs/.

66. LANG, Jean-Philippe. *Redmine Guide* [online]. [cit. 2023-11-26]. Dostupné z: https://www.redmine.org/guide.

67. CAMUNDA. *Camunda Modeler* [online]. [cit. 2023-11-30]. Dostupné z: https://camunda.com/platform/modeler/.

68. OBJECT MANAGEMENT GROUP, INC. *BPMN Specification* [online]. [cit. 2023-11-30]. Dostupné z: https://www.bpmn.org.

69. WALLACE, Evan; FIELD, Dylan. *Figma* [online]. [cit. 2023-11-16]. Dostupné z: https://www.figma.com.

70. ATLASSIAN. *JQL: the most flexible way to search Jira* [online]. [cit. 2023-11-06]. Dostupné z: https://www.atlassian.com/blog/jira-software/jql-the-most-flexible-way-to-search-jira-14.

71. SAINI, Rahul; MALI, Prashant. *How to get all issues of a project using Rest API* [online]. [cit. 2023-11-06]. Dostupné z: https://community.atlassian.com/t5/Jira-questions/How-to-get-all-issues-of-a-project-using-Rest-API/qaq-p/832316.

72. MEYER, Dave. *Log work via REST API for a given user* [online]. [cit. 2023-11-22]. Dostupné z: https://jira.atlassian.com/browse/JRACLOUD-30197.

73. TEMPO SOFTWARE. *Timesheets by Tempo - Jira Time Tracking* [online]. [cit. 2023-11-22]. Dostupné z: https://marketplace.atlassian.com/apps/6572/tempo-timesheets-jira-time-tracking?tab=overview&hosting=cloud.

74. ANONYMNÍ. *Webhook triggers in Redmine* [online]. [cit. 2023-11-27]. Dostupné z: https://www.redmine.org/issues/29664.

75. OBJECT MANAGEMENT GROUP, INC. *Unified Modeling Language* [online]. [cit. 2024-01-29]. Dostupné z: https://www.uml.org.

76. SCRUM.ORG. *Scrum* [online]. [cit. 2024-01-29]. Dostupné z: https://www.scrum.org.

77. ATLASSIAN. *Rate limiting* [online]. [cit. 2024-03-18]. Dostupné z: https://developer.atlassian.com/cloud/jira/platform/rate-limiting/.

78. POSTMAN, INC. *Postman API Platform* [online]. [cit. 2024-02-06]. Dostupné z: https://www.postman.com.

79. NGROK, INC. *ngrok* [online]. [cit. 2024-03-27]. Dostupné z: https://ngrok.com.

80. CAKE.COM INC. *Clockify* [online]. [cit. 2024-03-27]. Dostupné z: https://clockify.me.

81. GOOGLE. *Google Forms* [online]. [cit. 2024-04-02]. Dostupné z: https://www.google.com/forms/about/.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `src.txt` ..... odkaz na zdrojové kódy v GitHub
    - `test.ts` ..... testy Jira API
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `test` ..... záznamy z testování