FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Timer2Ticket II
Student:	Bc. Martin Paul
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem práce je dokončení již existujícího synchronizační nástroje (middlewaru) Timer2Ticket, jehož současnou verzi primárně vyvíjeli v rámci svých závěrečných prací Ing. Vít Štefan a Bc. Jakub Čermák tak, aby bylo možné její produkční nasazení.

Postupujte v těchto krocích:
1. Řádně prostudujte současnou verzi synchronizačního nástroje Timer2Ticket.
2. Zanalyzujte potřeby a záměry zadavatele se službou Timer2Ticket.
3. Na základě analýzy proveďte důkladný návrh potřebných změn a rozšíření tohoto synchronizačního nástroje.
4. Proveďte realizaci dle návrhu.
5. Navrhněte a realizujte vhodné sady testů s důrazem na bezproblémové budoucí použití a provoz služby.
6. Zhodnoťte dosažené výsledky a navrhněte vhodné budoucí směřování služby Timer2Ticket.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 18. května 2023 v Praze.

Diplomová práce

TIMER2TICKET II

Bc. Martin Paul

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Hunka
16. prosince 2023

Odkaz na tuto práci: Paul Martin. *Timer2Ticket II.* Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Rád bych poděkoval vedoucímu diplomové práce Ing. Jiřímu Hunkovi za odborné vedení, za pomoc a především za cenné rady při zpracovávání této práce. Dále bych chtěl poděkovat Ing. Oldřichovi Malcovi a Bc. Jakubovi Čermákovi za průběžné diskutování implementačních detailů a Bc. Maxi Hejdovi za pomoc s nasazením aplikace.
Jelikož se touto prací loučím s akademickou půdou, poděkování patří i celé Fakultě informačních technologií ČVUT v Praze, která mi poskytla drahocenné informace a znalosti. To vše bych však nezvládl bez podpory rodiny a přátel, kteří mi pomáhali, motivovali mě a vždy jsem u nich našel pochopení. Jmenovitě děkuji Soně Kerner za pomoc s překladem abstraktu.
Největší dík však patří osobě, díky které je život mnohem hezčí, za podporu nejen při psaní diplomové práce. Díky Ájo <3

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 16. prosince 2023 .....

Abstrakt

Tato diplomová práce se zabývá rozšířením, respektive úpravami nástroje Tiemr2Ticet, který synchronizuje časové záznamy mezi projektovými systémy a aplikacemi na sledování času. Práce navazuje na dvě stávající závěrečné práce.

Nejprve je analyzován současný stav nástroje Timer2Ticket s detailním popisem již vyvinutých komponent a částí nástroje. Analyzovány byly i požadavky zadavatele na budoucí funkcionality nástroje. Z analýzy poté vzejde seznam požadavků. Pro požadavky s nejvyšší prioritou bude proveden návrh, který bude upravovat a rozšiřovat současnou verzi nástroje. Následně dojde k realizaci tak, aby výsledná implementace mohla být nasazena v produkčním prostředí. Implementovaný systém byl podroben testům, které byly vybrány s důrazem na bezproblémové budoucí nasazení a provoz služby. V závěru práce je popsáno další možné rozšíření aplikace a její budoucí směřování.

Klíčová slova Timer2Ticket, aplikace pro sledování času, projektový systém, synchronizační middleware, middleware, JavaScript, TypeSrcipt, Vue.js, REST API, Auth0, Stripe, Fakturoid

Abstract

This thesis deals with the extension and modification of the Timer2Ticket tool, which synchronizes time records between project systems and time tracking applications. The thesis builds on two existing theses.

First, the current state of the Timer2Ticket tool is analysed with a detailed description of the already developed components and parts of the tool. The client's requirements for future functions of the tool were also analysed. The analysis then resulted in a list of requirements. Based on the priority of the requirements, modifications and extensions to the current version of the tool are outlined and subsequently implemented so that the resulting implementation can be deployed in a production environment. The implemented system has been subjected to tests, which have been selected with an emphasis on the smooth future deployment and operation of the service. The thesis concludes with a description of possible extensions to the application and its future.

Keywords Timer2Ticket, time tracking application, project management tool, synchronization middleware, middleware, JavaScript, TypeSrcipt, Vue.js, REST API, Auth0, Stripe, Fakturoid

Seznam zkratek

API	Application Programming Interface
BSON	Binary Javascript Object Notation
CMS	Content Management System
CRUD	Create, Read, Update, Delete
DIČ	Daňové identifikační číslo
DPH	Daň z přidané hodnoty
EU	Evropská unie
HMAC	Hash-Based Message Authentication Codes
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JQL	Jira Query Language
JWT	JSON Web Token
PDF	Portable Document Format
REST	Representational State Transfer
SHA	Secure Hash Algorithm
SQL	Structured Query Language
TESO	Time Entry Synced Object
TTL	Time To Live
T2T	Timer2Ticket
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

Úvod

Timer2Ticket je zamýšlena jako online aplikace, která by měla pomoci vývojářům či IT manažerům ušetřit čas nezáživné administrace při evidování času na projektech, respektive úkolech v rámci projektů, na kterých pracují. Myšlenka je taková, že by měl tento nástroj propojit obecně různé nástroje mezi sebou, aby nedocházelo ke zbytečnému přepisování dat z jednoho systému do druhého. Propojeny však budou především projektové nástroje a nástroje pro měření času, avšak obecně bude možné propojit dva projektové nástroje nebo nástroje pro měření času mezi sebou. Vývojář totiž obvykle pracuje na více projektech či úkolech v jeden den, a tak zadáváním a měřením stráveného času na konkrétních aktivitách může strávit nezanedbatelný čas, bez ohledu na možné chyby a nekonzistence, kterých se vývojář během zadávání času může dopustit. Uživatel by měl být od těchto zbytečných aktivit právě díky nástroji Timer2Ticket odstíněn.

Nástroje pro měření času slouží ke sledování času stráveného nad danou aktivitou, kterou uživatel provádí. Díky tomu pak může znát celkovou dobu strávenou prací na konkrétní aktivitě. Tyto nástroje mají obvykle co nejjednodušší ovládání, kdy je před zahájením práce zvolena aktivita, na které uživatel pracuje a poté zapne časomíru. Po dokončení práce časomíru zastaví, čímž se v aplikaci vytvoří příslušný časový záznam. Následně je možné zobrazit si přehled strávené doby na jednotlivých aktivitách. V oblasti IT se měření času používá především k lepším odhadům projektů, ať už z finanční nebo časové stránky [1]. Z tohoto důvodu jsou tyto nástroje hojně používány. Kromě toho mohou pomoci uživatelům s lepší organizací dne. Mezi tyto aplikace se řadí kupříkladu Toggl Track [2], Everhour [3], Timely [4] nebo TopTracker [5]. Některé aplikace nabízejí různé propojení s dalšími aplikacemi. Liší se také jejich zpoplatnění, kdy některé nabízejí několik tarifů předplatného, mnohé je však možné používat i zdarma.

Projektové nástroje naopak podporují plánování a organizaci projektů a umožňují sledovat průběh projektu, tedy zejména které úkoly jsou již na projektu hotové a které ještě ne. Jejich účel je podpora řízení projektu tak, že jsou k projektu vytvořeny požadavky, které se mohou nacházet v různých stavech, s různou prioritou a umožňují k nim přiřadit vykonavatele. Navíc je obvykle možné přímo ke konkrétnímu úkolu vykazovat strávený počet hodin. Umožňují však i další užitečné funkcionality, jako je například psaní popisu k požadavkům, diskuzi nad požadavky, přikládání příloh či další pokročilejší funkce, čímž může být řízení zdrojů nebo rozpočtu [6]. Příkladem je možno uvést následující aplikace: Jira [7], Redmine [8], JetBrains YouTrack [9], monday.com [10] nebo ClickUp [11]. I u těchto nástrojů nalezneme širokou škálu dodatečných funkcí s různou formou zpoplatnění, včetně aplikací zdarma.

Tyto nástroje budou prostřednictvím Timer2Ticket propojeny tak, aby se zaevidovaný čas strávený na projektech v jednom nástroji zpropagoval automaticky do ostatních. Uživatel tak pouze zvolí nástroje k synchronizaci, ty nakonfiguruje a následně naplánuje, jak často se údaje ze systémů budou synchronizovat. Tak nebude nutné vykazování provádět ve všech používaných nástrojích, ale pouze v jednom, jím preferovaným. Synchronizace však bude fungovat oboustranně, tedy změna v jednom systému se musí projevit v ostatních, jelikož některé úkony může být pro

uživatele příjemnější konat v jednom a jiné zase ve druhém nástroji. Tím budou veškeré vložené záznamy aktuální ve všech používaných nástrojích a dosáhne se úspora času při manuálním udržování aktuálnosti vložených údajů napříč systémy.

Prvotní implementaci provedl v rámci své diplomové práci Ing. Vít Štefan, který vytvořil backendovou část aplikace s jednoduchým grafickým rozhraním. Na jeho práci následně navázal Bc. Jakub Čermák, který navrhl nový frontend, včetně možné monetizace. Tento frontend však není napojen na původní backend.

Tato práce navazuje na obě závěrečné práce s cílem propojit obě části nástroje Timer2Ticket, které jsou na sobě zatím nezávislé a vhodně je rozšířit dle potřeb zadavatele. Bude detailně analyzován současný stav aplikace spolu s požadavky zadavatele, na jejichž základě bude navrženo rozšíření, respektive úpravy současné verze. Dle návrhu pak dojde k implementaci. Výstupem této diplomové práce bude nástroj Timer2Ticket, který je možné nasadit do produkčního prostředí.

Kapitola 1
Cíl práce

Hlavním cílem této diplomové práce je rozšířit již existující synchronizační nástroj (middleware) Timer2Ticket tak, aby následně bylo možné jeho nasazení v produkčním prostředí. Tento nástroj synchronizuje časové záznamy mezi projektovými nástroji a nástroji pro měření času.

Dílčími cíli je provedení důkladné analýzy již existujícího nástroje Timer2Ticket, na kterém již předtím pracovali v rámci své diplomové práce Ing. Vít Štefan a Bc. Jakub Čermák ve své bakalářské práci. Dále je pak nezbytné provést analýzu potřeb zadavatele, včetně jeho budoucích záměrů s nástrojem. Z analytické části vzejde seznam požadavků, kde bude jednotlivým požadavkům stanovena priorita a odhadnuta náročnost.

Dalším cílem bude návrh odpovídajících změn a rozšíření aplikace, dle kterého bude aplikace implementována. Návrh bude vypracován na základě vzešlých požadavků, avšak bude proveden pouze pro požadavky s nejvyšší prioritou. V této kapitole bude zahrnut popis změn v aplikaci a způsoby, jakým je realizovat tak, aby došlo k naplnění požadavků. Nebude chybět ani popis změny, respektive rozšíření databázového modelu a architektury, popis integrace se službami třetích stran nebo úprava již existujících případů užití.

Výstupem praktické části této práce bude vytvoření rozšířené, respektive pozměněné verze serverové (backend) části i uživatelského webového rozhraní (frontendu). V kapitole implementace bude popsáno, jak byly změny a rozšíření implementovány, včetně zdůraznění zajímavých částí, které bylo nutné vyřešit. Obsažena bude i část informující o možnostech změny konfigurace či o stavu nasazení nové verze systému. Tím bude splněn další dílčí cíl.

Implementované části budou podrobeny testům, což lze označit za další cíl této práce. Typy testů budou vybrány s důrazem na bezproblémové budoucí použití a provoz služby. U každého typu testů bude popsáno, jakým způsobem k němu bylo přistoupeno. Posledním cílem práce je určení možného rozvoje aplikace a její budoucí směřování v krátkodobém i dlouhodobém horizontu.

Kapitola 2
Analýza

Před zahájením prací na nástroji je nutné se s nástrojem nejprve seznámit, pochopit celkový kontext a porozumět, jak systém funguje. Proto je v této kapitole diplomové práce nejprve provedena analýza současného stavu nástroje Timer2Ticket, na jejž vznikly dvě závěrečné práce. Teprve poté bude možné touto prací na stávající verzi navazovat. Dále budou popsány oblasti, na kterých je nutné pracovat, aby mohlo dojít k úspěšnému nasazení aplikace do produkčního prostředí. Mimo to proběhne dále analýza potřeb, respektive záměrů zadavatele se službou Timer2Ticket. Na základě získaných informací bude navrženo rozšíření stávající verze nástroje Timer2Ticket.

2.1 Současný stav aplikace

Tato podkapitola se věnuje současnému stavu nástroje Timer2Ticket. Nejprve jsou pro přehled popsány obě závěrečné práce následované detailnějším popisem stávajících komponent a oblastí významných pro budoucí práci na tomto nástroji.

2.1.1 Diplomová práce Ing. Víta Štefana

Práce na nástroji Timer2Ticket byly započaty Ing. Vítem Štefanem, který položil základy této aplikace v rámci své diplomové práce [12] úspěšně obhájené v akademickém roce 2020/2021. V této práci autor nejprve nastínil důvody, proč by bylo vhodné tuto aplikaci vyvíjet. Následně analyzoval služby pro synchronizaci, z nichž vybral dva nástroje, které zakomponoval do základní implementace. Jednalo se o projektový nástroj Redmine a nástroj pro měření času Toggl. Volba pro tyto nástroje padla z důvodu využití ve společnosti Jagu s.r.o., se kterou bylo při vývoji tohoto nástroje spolupracováno.

Následuje rešerše již existujícího řešení. Autor zjišťuje, že existuje několik synchronizačních aplikací, služeb a doplňků pro výše zmíněné nástroje, avšak nikdy se nejedná o obecné řešení podporující více systémů. Pravděpodobně tedy neexistuje řešení, které by uspokojilo požadavky, které jsou na nástroj Timer2Ticket kladeny. Z tohoto důvodu tedy dává smysl tento nástroj vyvíjet, respektive navázat na již implementovanou část. [12]

Poté probíhal sběr a analýza požadavků, taktéž mezi zaměstnanci společnosti Jagu s.r.o. Výsledkem bylo zjištění udělat systém co nejjednodušší. Také jsou zde řešeny možnosti mapování objektů mezi Toggl Track a Redmine. Z této části práce vzešly konkrétní funkční a nefunkční požadavky, dle kterých byl nástroj Timer2Ticket implementován.

Po analýze probíhá návrh architektury aplikace, vnitřní implementace objektů včetně popisu synchronizace a nastavení služeb. Následuje popis perzistentního úložiště, zajištění bezpečnosti aplikace a na závěr i návrh frontendové části.

V kapitole realizace se autor věnoval především implementaci backendu, který byl doplněn o jednoduchého webového klienta. Zde popisuje, jakým způsobem byly jednotlivé části aplikace realizovány, včetně popisu struktury projektu a zdrojových kódů, zvolených technologií nebo řešení nastalých problémů. Na závěr je pak popsané nasazení systému a případné rozšíření o další synchronizační službu.

Následně byla naimplementovaná aplikace otestována – byly napsány unit testy a otestováno API. Poté proběhlo testování nad reálnými daty s testery, kterých bylo celkem pět.

Poslední kapitola se věnuje budoucnosti aplikace. Mimo možného rozvoje aplikace jsou zde navrženy i možnosti zpoplatnění aplikace.

Stav aplikace po dokončení diplomové práci Ing. Víta Štefana je takový, že aplikace byla nasazena a frontendová část je dostupná na URL `https://app.timer2ticket.com`. Zde si může kdokoliv založit účet a začít nástroj Timer2Ticket zdarma využívat.

Zdrojové kódy jsou pak dostupné v repozitářích na platformě GitHub [13]. Díky propojení s GitHub Actions je pak usnadněné nasazování nových verzí do produkce, jelikož je po sestavení vytvořen Docker image, který je následně nasazen v produkčním prostředí. Repozitáře jsou veřejné s licenčním ujednáním, které je v nich obsaženo [14]. Toto téma je však více popsáno v podkapitole Licence.

Do zdrojových kódů po dokončení diplomové práce zasahovali zaměstnanci společnosti Jagu s.r.o. Zde se však jednalo především o drobné úpravy a opravy, nešlo o vývoj či rozšíření funkcionalit aplikace. Detailnější popis a analýza všech implementovaných částí nástroje Timer2Ticket je obsažen níže v této podkapitole.

2.1.2 Bakalářská práce Bc. Jakuba Čermáka

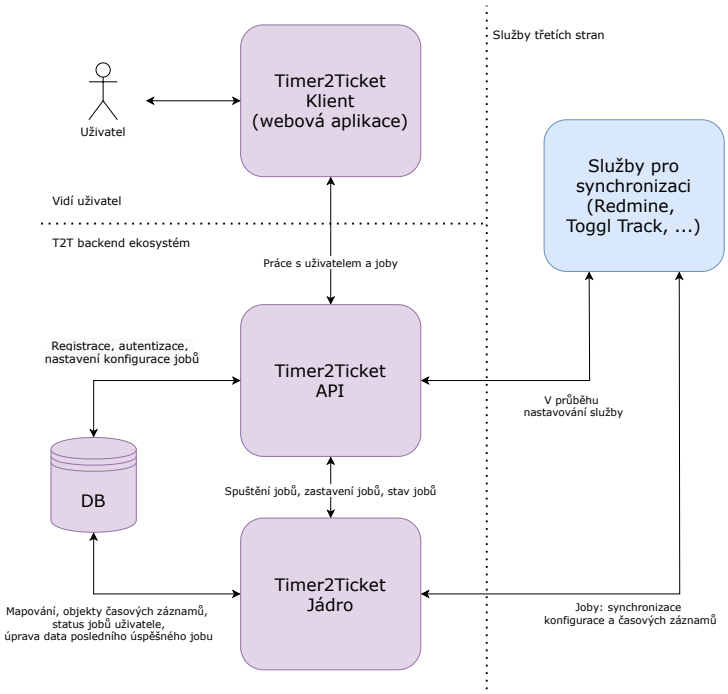
Na diplomovou práci plynule navázal Bc. Jakub Čermák, který svou bakalářskou práci [15] věnoval návrhu nového uživatelského rozhraní pro uživatele, včetně možnosti komerčního využití nebo propagačního webu. Vytvořená webová aplikace v budoucnu nahradí původního webového klienta. V oblasti návrhu uživatelského rozhraní využil Bc. Jakub Čermák možnosti spolupráce s dalšími studenty studujícími předmět Softwarový týmový projekt 2. Tato bakalářská práce byla zdařile obhájena v letním semestru akademického roku 2021/2022.

Stejně jako tato diplomová práce, i Bc. Jakub Čermák se nejprve věnoval analýze současného stavu nástroje Timer2Ticket. Zaměřil se však především na frontendovou část, tedy webovou stránku. Následuje analýza nástrojů, které Timer2Ticket integruje, respektive mohl by v budoucnu integrovat. Poté se věnuje způsobům spolupráce a formám vývoje. To z důvodu kooperace se dvěma studenty předmětu BI-SP2. Výstupem analýzy jsou pak funkční a nefunkční požadavky na jednotlivé části systému. Na závěr pak je ještě krátká zmínka o potenciálních zákaznících.

Následuje kapitola o volbě technologií. Bc. Jakub Čermák vybírá jako první nástroj pro tvorbu wireframů, poté framework a programovací jazyk pro frontend. Neopomnělo se ani na výběr pro tento nástroj nejvhodnější platební brány, která byla vybírána ze čtyř možností. Podstatným parametrem pro výběr byla znalost platební brány ve světě. Dále bylo v rozhodování zohledněno, o jaký typ brány se jedná a jak nákladné bude využívat danou bránu. V technických parametrech nebyly nalezeny výraznější rozdíly. Z analýzy vzešla nejlépe platební brána Stripe.

Kapitola návrh pojednává o tvorbě uživatelských členství a návrhu samotné webové stránky a propagačního webu. V kapitole implementace je popsáno využití Vue CLI a pluginů. Stejně tak jsou popsány problémy, které při implementaci nastaly a jejich vyřešení.

V kapitole testování je popsáno akceptační a usability testování. Testování použitelnosti bylo realizováno na vzorku čtyř testerů, všichni pocházejí z IT odvětví. V průběhu testování došlo ještě k několika změnám a návrhům, které by bylo vhodné v budoucnu doplnit. Na závěr bakalářské práce se autor zabývá možným budoucím rozvojem.



Obrázek 2.1 Schématické znázornění architektury nástroje Timer2Ticket, převzato z [12]

2.1.3 Backend

Jak již bylo zmíněno dříve, backendové části nástroje se věnoval ve své diplomové práci Ing. Vít Štefan. Z jeho návrhu vzešlo rozdělení serverové části aplikace na dvě části – tzv. T2T Core (jádro) část a T2T API část. Toto rozdělení bylo realizováno z důvodu oddělení logických celků do nezávislých částí, kdy T2T Core má na starosti synchronizaci dat mezi nakonfigurovanými nástroji. S T2T API pak komunikuje klient, jehož požadavky jsou zpracovávány. Těmito požadavky může být například registrace, autentizace, konfigurace nástrojů a úkony související se synchronizací – zjištění stavu nebo manuální zapnutí či vypnutí. Jak již plyne z názvu, tato část má vystavené rozhraní přístupné prostřednictvím REST. [12]

Pro snazší uchopení návrhu architektury jsou jednotlivé částí systému (včetně frontendové části a databáze) schématicky znázorněny na obrázku 2.1. Komponenty mezi sebou komunikují taktéž pomocí rozhraní REST. [12]

Pro implementaci backendu byl zvolen programovací jazyk TypeScript, který rozšiřuje JavaScript [16]. Jako běhové prostředí je využito Node.js [17], které využívá aplikační rámec Express.js při vytváření REST API [18]. Dále byly použity knihovny superagent (pro zasílání REST požadavků [19]), class-validator (pro validaci objektů [20]) a node-cron (pro plánování jobů [21]). [12]

T2T API

T2T API slouží jako prostředník mezi uživatelem (webovým klientem) a zbytkem aplikace, tedy T2T Core, databází a dalšími službami. Jedná se o REST API, na které jsou zasílány požadavky zvenčí. Požadavky jsou logicky rozděleny na 6 skupin – pro autentizaci, registrace, uživatele, joby, konfigurace synchronizovaných nástrojů a logy. Tato část následně komunikuje s T2T Core (například kvůli plánování synchronizace), avšak při konfiguraci nástrojů i s právě konfigurovanými systémy.

T2T Core

Hlavní náplní práce této komponenty je zpracovávání fronty jobů, ze které bude jednotlivé joby vyjímat a zpracovávat, bude-li ve frontě nějaký. Původní záměr byl takový, že při neúspěšném zpracování je job vrácen na konec fronty a bude zpracován později (do určitého počtu opakování), až na něj opět dojde řada. Implementováno je však jen jedno opakování, pokud tedy job není ani podruhé úspěšně zpracován, je pouze zalogována chyba. Poté je naplánováno další spuštění na základě prováděcího plánu. Tato část tedy komunikuje s nakonfigurovanými nástroji k synchronizaci. Dále je zde vystaveno REST API pro T2T API umožňující konfigurovat a plánovat synchronizaci. [12]

Autor diplomové práce původně navrhl tři druhy jobů. Nakonec však třetí, úklidový job, který měl mít za úkol odstraňování nepotřebných či nepoužívaných objektů, nebyl potřeba. Autor argumentuje tím, že kupříkladu Redmine již uzavřené úkoly neodesílá skrz API. Tím, že nebudou zaslány, tak budou odstraněny, což je jedno z popsaných chování systému v diplomové práci. [12] Nelze však vyloučit, že v budoucnu bude nutné tento job, nebo nějaký jiný, doimplementovat.

V implementaci jsou tak použity pouze dva synchronizační joby:

Job pro synchronizaci konfigurace

Tento job má na starosti objekty služeb, tedy především projekty a požadavky, a jejich synchronizaci napříč integrovanými nástroji. Při konfiguraci jsou objekty převedeny na objekty ostatních nástrojů. Následně je již kontrolován seznam mapování, kdy se ze všech služeb extrahují objekty a je sledováno, zdali došlo ke změně. V případě změny jsou objekty aktualizovány.

Existuje zde rozdělení nástrojů na primární a neprimární, kdy primární je jeden referenční nástroj zvolený uživatelem, ze kterého jsou objekty služeb mapovány do ostatních nástrojů. Při změně se tedy za „správné“ považují ty objekty, které jsou uloženy v primární službě.

Job pro synchronizaci časových záznamů

Jak vyplývá z názvu, hlavním účelem tohohle jobu je synchronizace vložených časových záznamů. Zde se nebere v potaz primární služba, pouze při mazání je podstatné, že je třeba záznam smazat ze služby, ve které byl vytvořen.

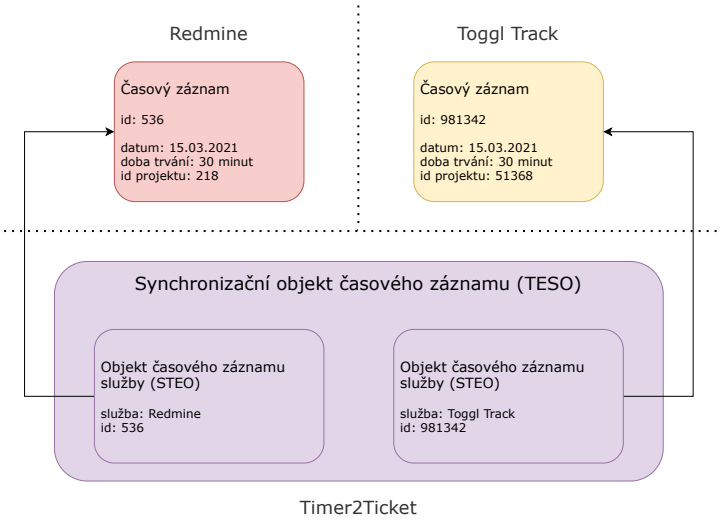
Po dokončení jobu je jeho výsledek zalogován do příslušné tabulky v databázi. Job se může nacházet vždy v jednom ze čtyř možných stavů. Spolu se stavem jobu je v logu obsažen i čas změny stavu. Logy jsou uchovávány pouze na 90 dní, následně jsou smazány. [12]

Pro uchovávání mapování objektů a časových záznamů mezi synchronizovanými nástroji je použito několik mapovacích objektů. Pro časové záznamy je použit Time Entry Synced Object (synchronizační objekt časového záznamu) reprezentující jeden časový záznam. Ten obsahuje kolekci Service Time Entry Object (objektů časového záznamu služby), který uchovává odkaz na instanci časového záznamu v daném nástroji k synchronizaci. Pro lepší představu těchto objektů je na obrázku 2.2 zobrazeno schéma mapování časových objektů. Pro ostatní objekty, které je třeba synchronizovat, jako jsou projekty nebo požadavky, je zde objekt Mapping spolu s kolekcí Mappings Object.

Integrace nástroje Redmine

Zatím jediný projektový nástroj, který je v nástroji Timer2Ticket integrován, je Redmine. Tato aplikace umožňuje získat některá uživatelská data prostřednictvím REST API, které je popsáno v oficiální dokumentaci. Toto API poskytuje základní CRUD operace a podporuje široce užívané formáty XML a JSON. Takto můžeme například upravovat, přidávat nebo mazat projekty a požadavky nebo vkládat časové záznamy. [22]

Většina požadavků, které lze prostřednictvím API vykonat, však vyžaduje autentizaci. Zde existují dvě možnosti, první je obecně používané zadání přihlašovacího jména a hesla, stejně



Obrázek 2.2 Schématické znázornění mapování časových záznamů, převzato z [12]

tak jako na webové stránce nástroje. Druhým pak může být použití API klíče, díky kterému není nutné vyplňovat přihlašovací údaje [22]. V rámci požadavku se pak klíč může vložit jako `key` parametr nebo do `X-Redmine-API-Key` HTTP hlavičky. API klíč je umožněno získat každému uživateli v sekci Můj účet, je-li to v nastavení povoleno. Právě pro využití API klíče se rozhodl Ing. Vít Štefan v rámci implementace [12]. Proto je tedy vyžadován od uživatele při konfiguraci.

Druhou věcí, která je vyžadována od uživatele, je přístupový bod, tedy základ adresy serveru, na kterém je Redmine nasazen. Poté je ještě nutné zvolit výchozí aktivitu, která bude použita v případě její nezvolení. [12]

Integrace nástroje Toggl Track

Toggl Track je druhý integrovaný software, tentokrát se však jedná o nástroj pro měření času. Tento nástroj má sice i placené plány s prémiovými funkcemi, avšak dle tvrzení Ing. Víta Štefana v diplomové práci je však dostatečný plán zdarma. Tento plán totiž umožňuje práci s projekty a štítky, které jsou pro Timer2Ticket zásadní. Štítky tak mohou být mapovány na aktivity z Redmine nebo štítky z Jiry. [12] Pokud chce uživatel v Toggl Track vložit nový časový záznam, vybere projekt nebo požadavek pomocí štítku, k němuž časový záznam náleží. Štítkem se zvolí i typ aktivity (v případě nezvolení se použije výchozí).

V tomto nástroji nalezneme workspaces. Jsou to oddělená prostředí, kde uživatelé mohou organizovat své projekty a týmy. Každá organizace tedy může mít svůj oddělený prostor na organizaci a zaznamenávání svých aktivit. Uživatel může náležet do více workspaces, kdy mezi jednotlivými workspaces může ve webovém prostředí přepínat. [23] Následně lze nalézt již intuitivnější projekty a štítky, neboli tagy. Projekty náleží do konkrétního workspace a jsou určeny k organizaci úkolů a zaznamenávání stráveného času na jednotlivých aktivitách. [24] K projektům je dokonce možné přidávat úkoly, tato vlastnost však v Timer2Tiket využita nebyla. K rozpoznání aktivit, kterým náleží časový záznam, je však použito tagů. [25]

I zde najdeme možnost přístupu k aplikaci prostřednictvím API, které poskytuje relativně velké možnosti interakce. K API nalezneme poměrně podrobnou dokumentaci. [26] V případě Timer2Ticket je podstatná práce s uživatelským účtem a náležejícími workspaces, projekty, tagy a časovými záznamy. Při konfiguraci této služby musí uživatel zadat nejprve API klíč, stejně jako v případě Redmine, a poté workspace, němuž náležející informace se budou synchronizovat.

Při prozkoumávání možností API vzbudily pozornost dvě skutečnosti, které je třeba uvést. Prvním je skutečnost, že již existuje nová verze API v9. Předchozí verze v8, použita Ing. Vítem

Štefanem v současné verzi implementaci, bude deaktivována od 1. dubna 2023. Povýšit bude nutné i Toggl Track Reports API z v2 na v3. Zde bude starší verze deaktivována 30. června 2023 [27]. Tím pádem bude povýšení verzí API v nástroji Timer2Ticket pravděpodobně jedna z priorit. [28]

Druhou věcí je podpora webhooků, tedy možnost automaticky reagovat na nějakou aktivitu v Toggl Track. V diplomové práci autor uvedl, že je Toggl Track nepodporuje [12]. To již neplatí, protože dle původního repositáře s dokumentací na Githubu byla první verze Toggl Webhooks API přidána dne 1. června 2022 [29]. Tomuto tématu se více věnuje podkapitola Webhooky v Toggl Track.

Plánování synchronizace

Plánování probíhá pomocí cron výrazů. Jedná se o textový řetězec, který definuje četnost provádění. Formát řetězce je posloupnost stanovených hodnot časových jednotek určující, kdy se má daný úkon vykonávat. Například řetězec `*/2 * * * * *` určuje vykonávání každou druhou sekundu. Více se o lze o formátu dočíst v dokumentaci knihovny node-cron [21] nebo v [30].

Plánování použitím node-cron se dosáhne tak, že se zavolá metoda `schedule` přijímající jako první parametr cron výraz a druhým je pak funkce, které se bude dle plánu vykonávat. Pro ukončení je možné danou naplánovanou úlohu ukončit zavoláním příslušné metody.

Cron výrazy spolu s knihovnou node-cron jsou ve zdrojovém kódu T2T Core využity pro čtení z fronty jobů, to se děje každých 10 sekund. Dále slouží k plánování všech typů jobů uživatele, kdy jsou joby vkládány do fronty jobů ke zpracování dle zvolené četnosti uživatelem, a jednou měsíčně pro mazání starých logů. [12]

Tyto plány jsou udržovány ve slovníku, ve kterém je klíč `userId`, tedy ID uživatele a hodnota naplánovaný úkon. Tomu je tak z důvodu, aby když uživatel pozastaví synchronizaci, bylo možné naplánovaný úkon ukončit.

2.1.4 Databáze

Pro ukládání dat se rozhodl Ing. Vít Štefan zvolit NoSQL dokumentovou databázi MongoDB. Tuto volbu obhajuje jednoduchostí, flexibilitou a snazší prací s uloženými JSON objekty při implementaci. [12]

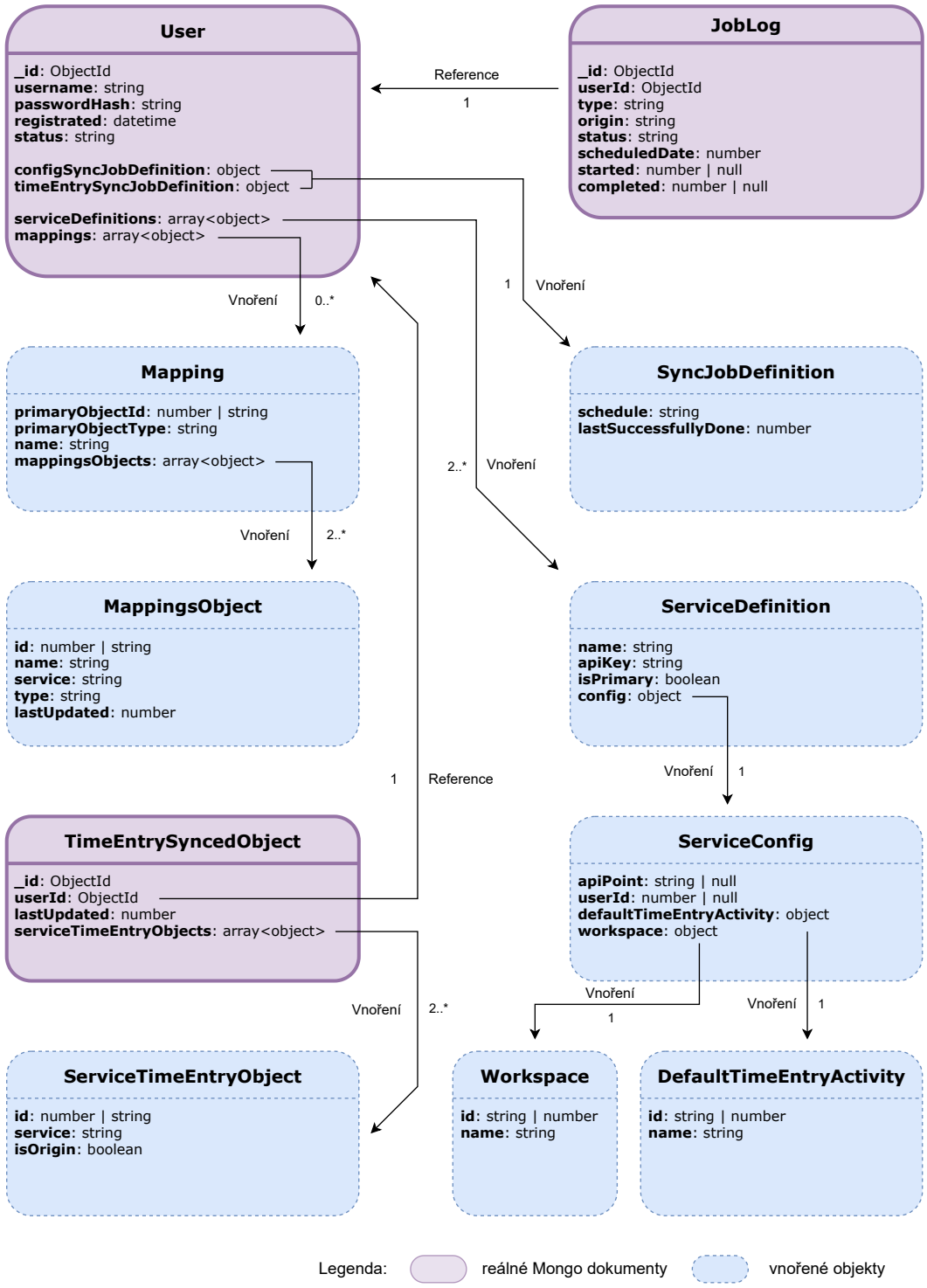
V této databázi jsou tři kolekce. Kolekce uživatelů obsahuje kromě přihlašovacích údajů i konfiguraci propojených nástrojů, mapování objektů služeb a informace o plánování jobů. Dále je obsažena kolekce pro mapování časových záznamů. Poslední kolekce pak slouží pro logy jobů. Schéma databáze pro Timer2Ticket je zobrazeno na obrázku 2.3.

Samotná databáze nekoná validaci ani autentizaci a autorizaci, je tedy třeba vkládat pouze validní záznamy. Validace vkládaných objektů probíhá v části T2T API, jenž následně komunikuje s T2T Core. Jelikož pouze T2T API nebo T2T Core komunikuje s databází, nemůže se stát, že je vložen objekt, který neprošel validací.

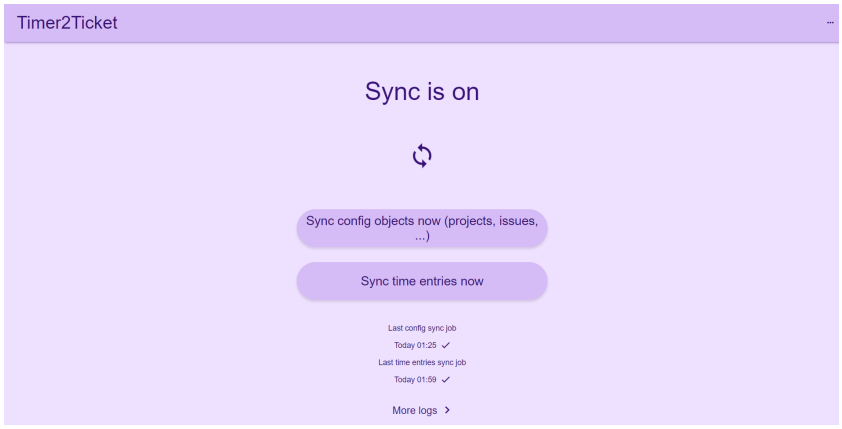
2.1.5 Frontend – stará verze

Jedná se o tzv. single-page webovou aplikaci, obsah stránky je tedy měněn dle akcí uživatele a nemusí se znovu načítat již jednou načtený HTML kód. Pro implementaci původní verze frontendové části byl zvolen taktéž TypeScript, respektive framework Angular [31]. Taktéž bylo použito RxJS Observable (pro sdílení dat [32]) a modul Angular Material pro grafické prvky. [12]

Data na stránce jsou zobrazena v jednoduchém sloupcovém rozložení. Web je stylizován do čistého fialového designu s cílem snadného ovládání. Po provedení některých akcí je uživateli v dolní části obrazovky zobrazena notifikační lišta obsahující informaci o výsledku akce. Tato lišta po uplynutí určitého času sama zmizí, případně je možné ji ručně skrýt.



Obrázek 2.3 Schéma databáze pro nástroj Timer2Ticket, převzato z [12]



Obrázek 2.4 Snímek obrazovky původní verze webové aplikace nástroje Timer2Ticket, z [33]

Technický popis

Webová stránka komunikuje především s backendem skrz část T2T API. Při komunikaci s API jsou všechny požadavky zasílány asynchronně. V některých případech je však nutné čekat na odpověď serveru, například během přihlašování uživatele. V tu chvíli je čekání znázorněno ztmavením obrazovky spolu s otáčejícím se kolečkem. Během konfigurace služeb k synchronizaci však komunikuje i s těmito službami, a to při ověření správnosti zadaného API klíče, nebo pokud aplikace potřebuje nějaká data z těchto systémů.

Prostřednictvím webu je uživateli umožněno nastavit nebo zobrazit si vše potřebné. Není proto překvapením, že možnosti prakticky reflektují vystavené API serverové části.

Ve frontendové části je třeba zamezit tomu, aby si návštěvník zobrazil stránky, které by si zobrazit neměl. Tedy kupříkladu zobrazení stránky konfigurace služeb nepřihlášeným uživatelem. Toho je dosaženo pomocí *guards*. Tím je například to, že nepřihlášený uživatel si může zobrazit pouze stránky přihlášení nebo registrace. Mimoto jsou *guards* využity při průchodu konfiguračním průvodcem, kde je ověřováno korektní vyplnění všech kroků. Dále dochází před odesláním HTTP dotazů pomocí *interceptors* k přidávání hlavičky s JWT a při přijetí odpovědi ke kontrole stavových kódů odpovědí. [12]

Popis grafického rozhraní

Po vstupu na webovou stránku je uživateli umožněno přihlásit se nebo registrovat, popřípadě požádat o změnu zapomenutého hesla. Přihlášení probíhá zadáním uživatelským jménem ve formě e-mailu a hesla. Po přihlášení se uživatel může v horní liště vždy odhlásit nebo si změnit heslo.

V případě, že ještě nemá nakonfigurované služby k synchronizaci, je vždy přesměrován na konfiguračního pětikrokového průvodce. V prvním kroku se vyberou služby k synchronizaci, které je nutné vybrat alespoň dvě. V této verzi tedy musíme zvolit Toggl Track a Redmine, žádné jiné zatím na výběr nejsou. Následuje konfigurace zvolených služeb, každá v samostatném kroku. Čtvrtým krokem je pak nastavení naplánování synchronizace objektů a časových záznamů. Posledním krokem je přehled nastavení a následné potvrzení.

Po potvrzení je uživatel přesměrován na stránku přehledu, která je zachycená na obrázku 2.4. Sem se taktéž dostane uživatel po přihlášení, má-li již služby nakonfigurované. V horní části lze spatřit informace o stavu synchronizace. Pod ní je možné manuálně spustit synchronizaci jak konfiguračních objektů, tak i časových záznamů.

Pod těmito tlačítky se nachází informace o posledních provedených jobech včetně času spuštění a stavu. Detailnější pohled na logy je možné získat po kliknutí na tlačítko More logs. Tím se otevře tabulka s detailním popisem posledních 100 spuštěných jobů. Jelikož se zobrazený stav

jednotlivých jobů může měnit, dochází zde k tzv. pollingu, tedy dotazování se serveru v pravidelných intervalech [12]. Tím je zajištěno, že v případě změny stavu je v uživatelském rozhraní vždy zobrazen aktuální stav.

V dolní části obrazovky se nachází pouze tlačítko umožňující vypnout nebo zapnout synchronizaci a možnost úpravy konfigurace. Uživateli je tak na stránce přehledu umožněno na webové stránce kontrolovat stav synchronizace včetně manuálního zapnutí, zapnout či vypnout synchronizaci, změnit konfiguraci synchronizovaných nástrojů a četnosti synchronizace nebo si zobrazit částečnou historii spuštěných jobů.

2.1.6 Frontend – Nová verze

Novou verzí webových stránek aplikace Timer2Ticket se zaobíral Bc. Jakub Čermák. Jednalo se o hlavní náplň jeho bakalářské práce. Pro realizaci byl zvolen JavaScriptový framework Vue.js [34], který byl vybrán na základě zkušeností autora, kvalitní dokumentace a všestranností tohoto aplikačního rámce. Oproti Ing. Vítu Štefanovi však nebyl zvolen TypeScript, ale JavaScript. [15]

Zde je však nutné upozornit, že se nejedná o plně funkční responsivní webovou aplikaci. Chybí zde například napojení na backend nebo integrace platební brány. Zobrazovaná data jsou tak pouze ukázková a v žádném případě nezobrazují reálné informace.

Technický popis

Projekt byl vytvořen s pomocí Vue CLI (Command Line Interface), což umožňuje počáteční konfiguraci projektu, ale také jeho správu či snadné přidávání pluginů do projektu [35].

Aplikace podporuje jazykové mutace, a to z důvodu, že necílí jen na české zákazníky. Toho je dosaženo použitím pluginu Vue I18n, který ve složce `locales` obsahuje jednotlivé soubory s jazykovými mutacemi. Výchozí jazyk je pak nastaven podle jazyka prohlížeče, ze kterého na web uživatel přistupuje. [15]

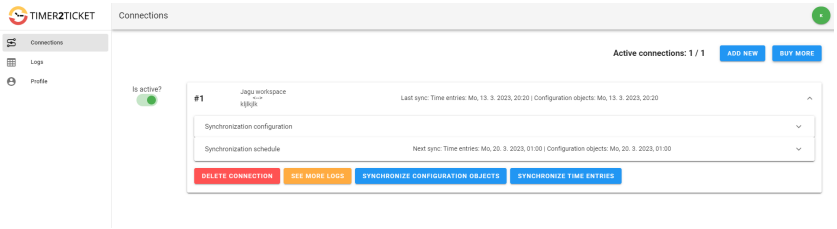
Pro směrování byl použit Vue Router. Tím je umožněno přecházet na jednotlivé stránky webu. Taktéž umožňuje ohlídat, aby se uživatel nedostal na stránku, kterou navštívit nemá. Dále je v aplikaci dosaženo sdílení dat mezi jednotlivými komponenty použitím pluginu Vuex. Jako grafická knihovna posloužila knihovna Vuetify, která je inspirována grafickým standardem Material Design a nabízí velké množství přizpůsobení grafických prvků. Knihovna umožňuje snadnou responsibilitu prvků nebo možnost zasílat notifikace. [15]

Pro tuto aplikaci byl stejně jako pro původní verzi aplikace (jak backend, tak i frontend) použit nástroj Sentry. Jedná se o software pro sledování chyb, který umožňuje logování a reporting chyb a další užitečné nástroje pro práci se zalogovanými chybami. Podporuje velké množství jazyků, včetně JavaScriptu. [15, 36]

Popis grafického rozhraní

Ze sběru požadavků vyplynulo, že bude potřeba celkem šest obrazovek, které sdružují pohromadě související funkce. Je nutné také zmínit, že se na základě stavu uživatele liší, které obrazovky si může zobrazit a jaké akce vykonat. Stavy jsou čtyři – nezaregistrovaný, přihlášený bez zvoleného členství, přihlášený se zvoleným členstvím a nepřihlášený. Podobně se ve stavech mohou nacházet spojení, tedy dvojice systémů k synchronizaci. Po nakonfigurování přejde spojení do stavu nakonfigurované, stane se aktivním a je spuštěna synchronizace. Pro tento úkon však uživatel musí mít zvolené členství. V případě vypnutí synchronizace se změní na neaktivní spojení. Spojení se může stát též neaktivním tím, že uživatelovi vyprší členství nebo si sníží počet aktivních spojení.

Obrázek 2.5 Navržené logo nástroje Timer2Ticket, převzato z [15]



Obrázek 2.6 Snímek obrazovky nové verze webové aplikace nástroje Timer2Ticket, z [15]

Po navštívení stránky je uživatel přesměrován na přihlašovací stránku, pro kterou byla využita platforma Auth0. Zde je uživateli umožněno přihlásit se (i přes sociální sítě), zaregistrovat se nebo si obnovit zapomenuté heslo.

Po úspěšném přihlášení se na základě stavu uživatele liší, co je zobrazeno dále. V případě, že ještě nemá zvolené členství, je vyzván k jeho zvolení. Jinou obrazovku mu není umožněno si zobrazit. Na obrazovce jsou přehledně v tabulce zobrazena jednotlivá členství, spolu s informacemi, co všechno zahrnují. Jelikož zatím není implementována platební brána, je tento krok přeskočen a po zvolení může uživatel pokračovat na stránku zobrazující spojení.

Na stránku spojení se dostane i přihlášený uživatel, který má zvolené členství. V této části je již funkční boční lišta. V její horní části se nachází logo, které vede na stránku spojení. Sem se lze stejně tak dostat i po kliknutí na tlačítko Connections v menu. Tlačítkem Logs lze zobrazit logy a Profile umožňuje provádět úkony s profilem.

Dále se na stránce nachází horní lišta, kde v levé části lze nalézt název sekce, ve které se uživatel nachází. V pravé části lze pak zobrazit bublinu s informacemi o právě přihlášeném uživateli. Z této bubliny se lze dostat do sekce správy účtu, změnit členství, přikoupit okamžité synchronizace nebo se odhlásit.

Tvorba loga byla přenechána grafikovi, který spolupracuje se společností Jagu s.r.o. Tím je Lukáš Olša. Výslednou podobu loga lze spatřit na obrázku 2.5.

Stránka spojení slouží ke správě spojení. V horní části lze spatřit počet aktivních spojení a maximální počet aktivních spojení. Dále zde najdeme tlačítko pro přidání dalšího spojení a možnost přejít na stránku členství, kde lze členství změnit nebo přikoupit okamžité synchronizace. Tuto obrazovku lze vidět na obrázku 2.6.

Níže lze spatřit vytvořená spojení, existují-li nějaká. Pro každé existuje box s informacemi o daném spojení. V levé části lze zapnout nebo vypnout synchronizaci (změna stavu na aktivní/neaktivní). Pokud se jedná o již nakonfigurované spojení, při zavřeném boxu lze v horní liště spatřit ID spojení, mezi čím probíhá synchronizace a informace o poslední synchronizaci. Po rozbalení si lze opět rozbalit informace o konfiguraci a plánování synchronizace daného spojení, které lze i editovat. V liště boxu pro plánování synchronizace se nachází zpráva o příští naplánované synchronizaci. V dolní části se pak nachází možnost smazat spojení, zobrazit si detailní informace o lozích a provést okamžitou synchronizaci.

Chceme-li vytvořit nové spojení, po stisknutí tlačítka se vytvoří nový box s otevřenou konfigurační částí. Zde je uživatel vyzván k vybrání služeb k synchronizaci a doplnění potřebných

informací. Uživatel si může ke každému vyžadovanému vstupu zobrazit nápovědu najetím kurzoru myši na ikonu otazníku vedle příslušného pole.

Stránka logy zobrazuje informace o uložených lozích z proběhlých nebo probíhajících jobů uživatele. Logy jsou zobrazeny formou tabulky. Jednotlivé sloupce obsahují ID spojení, jaké služby spojení propojuje, kdy byl job naplánován a dokončen, typ jobu, původce (manuální nebo automatické spuštění) a výsledek, tedy zdali job doběhl korektně či nikoliv. Výsledky lze podle jednotlivých sloupců filtrovat nebo seřadit. U každého záznamu (tedy řádku v tabulce) si lze rozkliknout další informace o daném jobu, kdy se zobrazí komentář. Zde například bude chybová hláška, nedoběhl-li job bezchybně.

Poslední stránkou je pak profil. Tato sekce je však rozdělená na tři podsekce, mezi nimiž lze přecházet pomocí menu umístěného v horní liště. První podsekcí, do níž se uživatel dostane po kliknutí na tlačítko Profile v hlavním menu, je podsekce Settings, tedy nastavení. Zde může uživatel vidět svůj e-mail, poté změnit heslo, nastavit si zasílání notifikací, zvolit časovou zónu nebo vložit kupón, má-li nějaký.

Výběr časové zóny je v aplikaci proto, aby se zobrazovaly správné časové údaje, tedy v časové zóně uživatele. Aplikace totiž necílí jen na české uživatele. Obzvláště důležité může být zvolení časové zóny v případě nastavení času synchronizace. Volbou časové zóny tak lze dosáhnout synchronizace ve správný čas. [15]

Další podsekcí je pak členství, neboli Membership. V horní části je zobrazeno současné členství uživatele s informací, do kdy je platné. Vedle se nachází tlačítko s možností přerušení předplatného.

Níže na této stránce se nacházejí tři rozbalovací boxy. První box umožňuje změnit členství. Zobrazena je zde stejná tabulka, jako je tomu při volbě členství při registraci, rozšířená o možnost přikoupení dalších spojení nad rámec zvoleného členství. Při provedení změny je v boxu zobrazena celková vypočítaná částka. Stejná informace se nachází i v dolní části obrazovky, kde je zobrazena jak celková cena k zaplacení nyní, tak i pravidelná cena v budoucích měsících. Některé položky lze totiž koupit jednorázově, viz další odstavce.

Další box umožňuje nákup okamžitých synchronizací. Ty je v současném stavu možné přidávat po dvaceti, kdy za každou dvacítku okamžitých spojení jsou účtovány jednorázově čtyři americké dolary. Poslední box obsahuje platební údaje.

Dokončil-li uživatel objednávku, po kliknutí na tlačítko See summary v pravém dolním rohu se zobrazí shrnutí objednávky, ze kterého se lze vrátit na původní obrazovku. Pokud je s objednanými položkami spokojen, stisknutím tlačítka Pay přejde k platební bráně (která zatím není naimplementovaná).

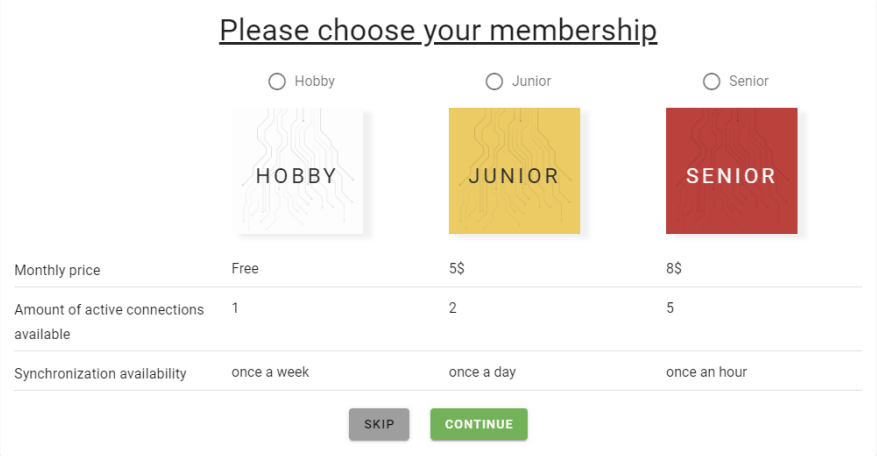
Poslední sekce nese název Billing – platby. V horní části je možné nastavit platební údaje. Níže se nacházejí provedené objednávky – zobrazeno je číslo objednávky, datum, cena a možnost stáhnout fakturu.

2.1.7 Propagační web

Při sběru požadavků ve své bakalářské práci dospěl Bc. Jakub Čermák ke zjištění, že bude nutné vytvořit propagační web. Web má nabízet základní informace, přehled funkcionalit a možností nástroje Timer2Ticket a informovat o nabízených členstvích. Z tohoto webu se bude možné přihlásit, respektive zaregistrovat.

Pro uskutečnění tohoto webu byl zvolen systém October CMS. CMS (systém pro správu obsahu) umožňuje snadný a rychlý vývoj webových stránek. October CMS nabízí například vytváření a upravování šablon, optimalizaci pro vyhledávače a kvalitní dokumentaci. [37]

Na návrhu propagačního webu pracoval Benedek Molnár v rámci předmětu BI-SP2, který pro tento úkol využil nástroj pro návrh uživatelského rozhraní Figma. Navržen byl jednostránkový web rozdělený do několika tématických částí. V horní části se nachází menu spolu s tlačítky pro přihlášení nebo registraci. Menu odkazuje do jednotlivých sekcí umístěných níže na stránce.



Obrázek 2.7 Přehled navrhovaných uživatelských členství, snímek obrazovky z nové verze frontendu [15]

První sekce cílí na upoutání uživatele. Nachází se zde popisek aplikace a propagační video popisující hlavní funkce a výhody používání. Video však zatím neexistuje. Součástí sekce je i tlačítko informující o členství zdarma.

Druhá sekce nabízí návštěvníkovi webu výčet nejpodstatnějších funkcí, které Timer2Ticket nabízí. Poté jsou přehledně vyzobrazeny jednotlivé členství aplikace, které lze mezi sebou snadno porovnat. Nebude chybět ani tlačítko pro kontaktování provozovatele ohledně poptání množstevní slevy.

Čtvrtá sekce představuje projekt jakožto open-source. Je zde kupříkladu odkaz na dokumentaci projektu, informace o možném budoucím rozvoji aplikace či o možnosti lokálního nasazení uživatelem. Taktéž se zde nachází tlačítko umožňující finanční podporu projektu.

Předposlední sekce poskytuje detailní informace o projektu, včetně jeho historie a vývojářích, kteří se podíleli na vývoji. Nacházet se zde budou i informace o budoucích plánech. Poslední, šestá sekce, pak obsahuje kontaktní formulář.

Propagační web lze nalézt na adrese `timer2ticket.com`. Na stejnou stránku se lze též dostat přes adresu `timer2ticket.cz`. V době citace se zde však nachází webová stránka, která byla realizována v rámci předmětu BI-SP2, kdy se jedná o nedokončenou verzi, které chybí několik navrhovaných sekcí. [38]

2.1.8 Monetizace

Hlavním zdrojem příjmů by měly být uživatelské členství. Členství si vybere každý uživatel před používáním nástroje Timer2Ticket. V rámci bakalářské práce Bc. Jakuba Čermáka byly navrženy tři typy členství, které se liší v ceně, maximálním množství aktivních spojení a v pravidelnosti synchronizace. Nechybí ani členství zdarma poskytující jedno aktivní spojení s možností synchronizace jednou týdně. Přehled navrhovaných členství je vyobrazen na obrázku 2.7. V případě nevyhovujícího počtu spojení bude dále možné dokoupit jednotlivá aktivní spojení.

Další možností výdělku bude prodej okamžitých synchronizací. Ty umožňují uživateli okamžitě synchronizovat danou dvojici synchronizovaných nástrojů, tedy i častěji, než umožňuje zvolené členství. Poslední možností je podpora vývojářů jednorázovým příspěvkem. Dárci by poté jako forma poděkování přišel kupón, který po zadání v nástroji Timer2Ticket uživateli poskytne okamžité synchronizace v závislosti na výši příspěvku.

2.1.9 Bezpečnost

V původní verzi nástroje vyvinutém v rámci diplomové práce Ing. Vítem Štefanem je použit JWT, který je uživateli vygenerován po úspěšné autentizaci a následně slouží k autentizaci při používání webové aplikace (nebo API). Toto je použito pouze v backendové části aplikace v části T2T API. V části T2T Core není potřeba, jelikož s touto částí bude komunikovat pouze s T2T API, ve které již je uživatel autentizován.

V části T2T API existují endpointy pro autentizaci a registraci uživatele. Část pro autentizaci obsahuje tři POST požadavky. První pro přihlášení při zadání uživatelského jména a hesla v případě správně zadaných údajů vrátí vygenerovaný JWT. Zbylé dva slouží pro obnovu hesla – jeden vygeneruje TTL token platný jednu hodinu. Druhý pak přijímá tento token spolu s novým uživatelským heslem, které je uživateli nastaveno.

Pro registraci uživatele existují dva endpointy, taktéž POST. Slouží k zaslání TTL tokenu pro dokončení registrace uživatele e-mailem a k následnému potvrzení registrace.

V návrhu nového frontendu se však Bc. Jakub Čermák rozhodl využít platformu Auth0. Ta již řeší možnost přihlášení a registrace uživatele (včetně přístupu přes sociální sítě), i obnovení zapomenutého hesla. Platforma nabízí několik plánů, včetně plánu zdarma, který umožňuje mít až 7000 aktivních uživatelů s neomezeným počtem přihlášení [39].

Vzhledem k rozdílným technologiím použitým na backendu a novém frontendu bude nutné sjednotit, jakým způsoben autentizovat a autorizovat uživatele. Výběr bude rozhodnut na základě analýzy dále v textu.

2.1.10 Licence

Text licence zdrojových kódů nástroje Timer2Ticket je k dispozici na GitHubu v repozitáři obou backendových částí [14]. Licence umožňuje používání, kopírování, publikování a distribuování kopií zdrojových kódů. Umožněno je taktéž použití pro komerční účely, avšak program nelze prodávat ani pronajímat třetím stranám.

Výše uvedené je však umožněno pouze se zachováním souboru s licencí ve všech vytvořených kopiích softwaru a je nutné uvést autorství autora licence. Přispěním do zdrojových kódů tohoto softwaru je dle licence vývojář povinen přidat je do oficiálního úložiště.

Po diskuzi se zadavatelem práce se předpokládá, že tato licence bude zachována a tak zdrojové kódy open-source aplikace budou dále k dispozici veřejnosti.

2.1.11 Směr budoucího vývoje

Je zřejmé, že aplikace nabízí mnoho směrů, kterými ji lze rozvíjet. Může se jednat jak o zefektivnění služby nebo rozšíření o další nástroje k synchronizaci, tak i například její monetizaci. Možnostem rozšíření aplikace a jejího budoucího rozvoje se věnovali autoři obou závěrečných prací. Ty nejzajímavější jsou popsané v rámci této sekce, detailnější popis pak nabízí podkapitola Požadavky.

Ing. Vít Štefan si všímá možné neefektivity při synchronizování časových záznamů. Počet časových záznamů bude průběžně narůstat, což zvýší náročnost jak paměťovou na straně Timer2Ticket, tak především při synchronizaci, kdy se musí všechny záznamy zesynchronizovat. Jednou možností by mohlo být synchronizovat ty, u kterých proběhla změna. Kupříkladu Toggl Track API již umí vyfiltrovat záznamy dle času modifikace. Druhou možností je pak použití webhooků, tedy zasílání událostí při akci nad daným objektem. Tím by bylo možné držet objekty aktuální téměř v reálném čase, jelikož by synchronizace změny proběhla téměř okamžitě. Zde je však nejprve třeba provést analýzu, zdali podporované nástroje umožňují použití webhooků.

Podobný problém nastává i u synchronizace konfiguračních objektů. Řešení by mohlo být obdobné, jako je tomu u synchronizace časových záznamů.

Další věcí, kterou by stálo za to do Timer2Ticket doplnit, je vylepšení logů. Uživatel by jistě ocenil, kdyby v případě chyby aplikace zobrazila, kde při synchronizaci nastala chyba. V neposlední řadě pak bude možné aplikaci rozšiřovat co se podporovaných nástrojů týče. Zde jsou možnosti rozvoje prakticky neomezené.

Po dokončení bakalářské práce navrhl Bc. Jakub Čermák, čím se zabývat v budoucnu. Všechny návrhy mají vytvořené úkoly v projektovém nástroji Redmine. Celkem zde vytvořil 48 úkolů s detailním popisem a návrhem řešení. Mimo jiné jim stanovil i prioritu a odhad náročnosti. Mezi priority řadí Bc. Jakub Čermák vytvořit verzi pro nekomerční použití, napojení platební brány, napojení na backend a přidání dalších nástrojů k synchronizaci. [15]

2.2 Sběr požadavků vyplývající ze současného stavu aplikace

Autoři předchozích prací nastínili možné směřování a budoucí vývoj nástroje Timer2Ticket. Před implementací je však u některých návrhů třeba nejprve podrobit analýze možnosti a technologie, které by mohly být následně využity.

2.2.1 Použití Webhooků

Jelikož by implementace webhooků pomohla zefektivnit synchronizaci objektů mezi službami, je vhodné ověřit, zdali by mělo smysl upravit implementaci současné verze Timer2Ticket a webhooků při synchronizaci použít. Tím by se při běhu konfiguračního jobu nemusely kontrolovat všechny synchronizované objekty, zdali u nich došlo ke změně, ale server by byl automaticky na změnu upozorněn.

Použitím webhooků by také mohlo dojít k rozšíření obchodního modelu o další členství, který by právě díky webhookům podporoval okamžité synchronizace (za předpokladu, že daný nástroj webhooky podporuje).

Webhooky v Toggl Track

V nástroji Toggl Track webhooky umožňují odebírat zprávy při nastalých událostech. Díky filtrům je navíc možné odebírat pouze omezený rozsah událostí. Toho lze dosáhnout použitím klíče `event_filters` v těle požadavku ve formátu JSON. Hodnotou je pak pole aplikovaných filtrů, kde každý filtr obsahuje `entity` – entitu, které se filtr týká a `action` – notifikace proběhne po stanovené akce.

Akcí může být vytvoření, úprava nebo smazání. Entitou jsou mimo jiné i workspaces, projekty, štítky, časové záznamy a uživatel, tedy všechny, se kterými vyžaduje Timer2Ticket interagovat. Je tedy velmi pravděpodobné, že by se webhooky daly v implementaci využít a tím zefektivnit synchronizaci alespoň na straně nástroje Toggl Track. [26]

Následující ukázkový filtr tak odebírá události týkající se pouze úpravy časových záznamů. V obou případech je možné nahradit hodnotou `*`, jedná se o zástupný symbol připouštějící všechny možnosti. [26]

```
"event_filters": [
    {
        "entity": "time_entry",
        "action": "updated"
    }
]
```

Pomocí API je možné webhooky přidávat, upravovat a odebírat. Pro konfiguraci webhooku je ještě podstatné `workspace_id`, tedy identifikátor daného workspace, a URL, na které se budou informace o vykonaných událostech zasílat.

Pro úspěšné vytvoření webhooku je zapotřebí ověřit URL adresu na straně serveru odebírající události. To je nutné provést z důvodu ověření, že má uživatel přístup k událostem. Validace proběhne tak, že se zašle HTTP GET požadavek na adresu spolu s path parametry `workspace_id`, `subscription_id` a `validation_code`, kde `subscription_id` a `validation_code` je získáno v první odeslané události na server po vytvoření webhooku.

Další důležitou věcí, kterou umožňuje webhook API, je ověřování přijatých událostí. Ověřování je důležité z toho důvodu, abychom se ujistili, že jsou na URL adresu serveru odesílány data z aplikace Toggl Track, nikoliv potenciální záškodník. Ověření dosáhneme díky HTTP hlavičce `X-Webhook-Signature-256`, která má tvar `sha256={value}`, kde `value` je HMAC hash založený na šifrovacím algoritmu SHA-256. Jako vstup pro výpočet hashe je použit obsah těla příchozí události. Tajným šifrovacím klíčem je pak tzv. secret, který lze buďto zadat při vytváření webhooku nebo následně upravit v těle zprávy atributu `secret`, v případě vynechání atributu je pak hodnota vygenerována automaticky. [26]

Webhooky v Redmine

Dle průvodce pro vývojáře Redmine neposkytuje možnost webhooky, respektive o této metodě zde není žádná zmínka [40]. Je zde však článek o Redmine plugin hooks [41].

Zde se lze dozvědět, že Redmine podporuje koncept hooků, avšak přiložený seznam možností je poměrně omezený. Poskytuje například možnost reakce na vytvoření nebo editace požadavku, avšak pro manipulaci s projektem zde hook není.

Nejvýraznějším problémem však je, že hooky nelze vytvářet a spravovat přes REST API, jako je tomu u ostatních zde analyzovaných nástrojů. Pro využívání hooků by bylo nutné napsat vlastní plugin. To může být pro potenciální uživatele velmi omezující, jelikož by již nástroj nebyl Timer2Ticket tak přímočarý. Za předpokladu, že by byl plugin rozšiřující Redmine o webhooky napsán, by tak uživatelům nestačilo pouze vytvořit a nakonfigurovat spojení ve webové aplikaci Timer2Ticket, ale nejprve by museli rozšířit svůj Redmine o tento plugin. To koliduje s myšlenkou jednoduchosti a přímočarosti používání Timer2Ticket.

Na webu lze najít již vytvořené pluginy pro Redmine implementující webhooky, například [42] nebo [43], nicméně umožňují reagovat pouze na operace prováděné s požadavky. Jak bylo zmíněno dříve, to je pro potřeby Timer2Ticket nedostatečné.

Z výše uvedených důvodů bylo tedy rozhodnuto integraci Redmine pomocí webhooků v rámci této diplomové práce neimplementovat. Tato myšlenku však zůstává otevřená jako možné budoucí vylepšení aplikace.

Webhooky v Jira

I nástroj Jira podporuje webhooky. Lze je vytvářet přes webové rozhraní aplikace nebo přes REST API, které je relevantní pro integraci v nástroji Timer2Ticket. V těle volání je specifikováno jméno webhooku, URL, na kterou budou události zasílány, jaké události mají být zasílány, filtr a zdali má být do události vloženo tělo zprávy.

Webhooky umožňují reagovat na velké množství situací. Podstatné pro Timer2Ticket je, že umožňují reagovat na vytváření, úpravu a mazání požadavků, projektů a časových záznamů. Zachytávat však lze i veškeré nastalé události, které webhooky podporují.

Zachytávané události je možné i filtrovat, a to parametrem `filters` v těle zprávy při vytváření webhooku. Pro filtrování je využit již zmíněný jazyk JQL. [44]

Příchozí zprávy na server neobsahují žádnou možnost autentizace, že původcem zprávy je skutečně Jira. Proto bude nutné každou zprávu o změně zkontrolovat. Nejjednodušší je pravděpodobně získat modifikovaný objekt přes REST API a poté pracovat s tímto objektem, ne

s objektem obdrženým přes webhooky. Tím se zajistí, že se do systému nedostanou falešná data od potenciálních útočníků.

2.2.2 Sjednocení autentizace a autorizace

Dále bude třeba rozhodnout, jakou platformu zvolit. V původní verzi bylo použito JWT s knihovnou node-jsonwebtoken [45], v realizaci nového frontendu je však použit Auth0.

JWT

Autentizace a autorizace je dosáhnuto pomocí volání knihovních metod. Přihlášení proběhne pomocí metody `sign`, do které se v případě implementace T2T API vkládá uživatelské jméno a doba expirace tokenu. Navrácen je JWT token, který je uložen na straně klienta.

Tento token se poté používá k autentizaci volání, kdy je manuálně vkládán do příslušné hlavičky. Na straně serveru dochází k jeho ověření pomocí metody `verify`.

Výhoda tohoto přístupu je spatřena v možnosti plné kontroly nad autentizací a autorizací, kdy se pouze volají výše zmíněné metody ve zdrojovém kódu aplikace. Dále se jedná o poměrně jednoduchý a přímočarý způsob autentizace a autorizace.

Bohužel se k využití JWT a zmíněné knihovny váží i nevýhody. Jednou z nich je nutnost vlastní implementace správy uživatelského účtu – změna a obnova zapomenutého hesla. Dalším problémem pak může být možnost přihlášení pomocí sociálních sítí, které je jedním z požadavků nového frontendu. Toto by bylo nutné řešit přímo ve zdrojovém kódu aplikace.

Auth0

Auth0 je cloudová platforma pro správu identity a přístupu, která umožňuje do aplikací snadno přidávat autentizaci a autorizaci uživatelů. Platforma nabízí širokou škálu funkcí, mimo jiné možnost přihlášení přes sociální sítě, vícefaktorovou autentizaci nebo správu uživatelů a rolí, včetně možnosti přizpůsobení mnohého. Pro Timer2Ticket je podstatné především autorizace a autentizace uživatelů spolu se správou účtu – obnovení a změna hesla. To vše Auth0 poskytuje. [39]

Mezi její výhody patří snadná integrace pomocí vystaveného API, bezpečnost například díky možnosti vícefaktorového ověření nebo kvalitní dokumentace. Nevýhodou pak může být cena za využívání platformy (tuto stránku věci rozebíral ve své práci Bc. Jakub Čermák [15]) nebo závislost na třetí straně. [39]

Veškerou správu lze vykonat ve webovém klientovi. Po registraci uživatelského účtu je nabídnuto vytvoření aplikace včetně detailního nastavení vlastností, jako je logo aplikace, titulek nebo nastavení adres URL. Dále lze vybrat databázi uživatelů či možnosti přihlášení přes sociální sítě.

Platforma také poskytuje široké možnosti přizpůsobení přihlašovací obrazovky. Možné je měnit logo, barvy, fonty, rámečky či pozadí. Možné je i zasahovat přímo do zdrojového kódu HTML. Mimo to je možné i měnit e-mailové šablony, které jsou zasílány pro potvrzení registrace či obnovení hesla.

Auth0 umožňuje využít i vlastní databázi. Tu lze synchronizovat s použitím tzv. akčních skriptů, tedy skriptů, které se vykonají po konkrétní akci. Těchto akcí je šest: přihlášení, vytvoření, verifikace účtu, změna hesla, získání uživatele a smazání.

Ve webovém klientovi lze též zobrazit registrované uživatele včetně dalších informací. Těmi může být například poslední přihlášení, použitý prohlížeč nebo poskytovatel identity. Zde je též možné uživatelům přidělovat role či oprávnění.

Propojení s aplikací je snadné díky volbě typu aplikace, kdy jsou po zvolení nabídnuty podporované jazyky a technologie s detailním návodem, jak integrovat Auth0. Na výběr je mimo jiné i JavaScript, Angular nebo Vue, tedy technologie použité při implementaci starého či nového frontendu.

Keycloak

Další platforma pro správu identity a přístupu, jejíž použití připadá v úvahu, je Keycloak. S touto platformou má zkušenosti vedoucí práce, který ji zmínil na jedné z konzultací. Z tohoto důvodu bylo rozhodnuto zvážit její použití a prozkoumat možnosti, které tato platforma nabízí.

Jedná se o open-source platformu, která poskytuje podobné funkce jako Auth0. Taktéž nabízí možnost přihlášení přes sociální sítě, vícefaktorovou autentizaci nebo správu uživatelů a rolí. Přesto však lze najít oblasti, ve kterých se obě platformy velmi liší. [46]

Největší výhoda Keycloaku je spatřena v již zmíněném open-source. Keycloak je tedy dostupný zdarma, navíc má aktivní komunitu vývojářů, kteří platformu dále rozvíjejí. Keycloak taktéž poskytuje poměrně rozsáhlé API. Nevýhodou naopak může být oproti Auth0 složitější konfigurace a nastavení nebo menší podpora. Keycloak taktéž není cloudová platforma, je tedy potřeba mít aplikaci nasazenou na vlastním serveru.

Po spuštění serveru (Keycloak nabízí i image pro Docker) lze přejít do webového klienta, ve kterém lze Keycloak nakonfigurovat dle vlastních potřeb. Pro použití je potřeba nejprve vytvořit tzv. realm, což je oddělený prostor pro správu uživatelů, rolí a klientů. V realmu je třeba následně vytvořit klienta reprezentující aplikaci na Keycloak serveru. Konfigurace klienta poskytuje širokou škálu možností, podstatná je konfigurace adres URL a URI pro přesměrovávání. [46]

Ve webovém klientovi platformy lze následně zobrazit přehled uživatelů, taktéž s podrobnými informacemi o uživateli. Také je možné vytvářet role a uživatelům je přiřazovat. Platforma rovněž umožňuje zvolit, zdali má být použito pro přihlašování uživatelské jméno či e-mail, zdali je možné uživatelské jméno měnit či jestli je možné mít více účtů na jeden e-mail.

Keycloak nabízí velké možnosti přizpůsobení vzhledu přihlašovací obrazovky. Lze zvolit, co vše se má na stránce zobrazit – možnost registrace uživatelů, obnovy hesla či zapamatování přihlášení. Změnit vzhled stránky je rovněž možné, nikoliv však přes grafický editor, je třeba upravit a doplnit příslušné soubory. Postup, jak přihlašovací stránku upravit dle svých potřeb, je popsán například v [47].

Vzhledem k tomu, že Keycloak je nutné mít spuštěný na vlastním serveru, je s jeho provozem spojeno ještě několik dalších věcí. Pro ukládání dat v produkčním prostředí je třeba mít relační databázi. Podporováno je pět databázových systémů, mimo jiné MySQL nebo PostgreSQL [46]. Dále je třeba mít SMTP server pro zasílání ověřovacích e-mailů nebo e-mailů pro obnovu hesla. Z těchto důvodů stojí za zvážení, zdali se vývojáři vyplatí provozovat a udržovat infrastrukturu potřebnou pro provoz Keycloaku, či upřednostnit komerční řešení, jako výše zmíněnou platformu Auth0.

Výběr autorizační a autentizační platformy

Po konzultaci se zadavatelem práce a Oldřichem Malcem, projektovým manažerem společnosti Jagu, byla upřednostněna platforma Auth0. To, že se jedná o cloudovou platformu s velmi jednoduchou správou a konfigurací, bylo významným argumentem podporující tuto volbu. Nebude totiž potřeba údržba další části aplikace, což by bylo v případě použití platformy Keycloak potřeba. Auth0 navíc podporuje přihlášení přes sociální sítě, což bylo mimo jiné jedním z požadavků v bakalářské práci Bc. Jakuba Čermáka [15].

Další výhodou je snadná změna v případě nasazení open-source verze aplikace na vlastním serveru. Pro použití vlastního účtu platformy Auth0 stačí pouze změnit několik proměnných prostředí při konfiguraci nástroje Timer2Ticket, což jen podporuje snadné použití aplikace v lokálním prostřední.

2.2.3 Volba platební brány

Jak již bylo zmíněno, platební brány již byly analyzovány v bakalářské práci Bc. Jakuba Čermáka [15], který vybral platební bránu Stripe. O platební bráně Stripe hovořil na jedné

z konzultací i Ing. Oldřich Malec. Ten má již s integrací platební brány zkušenosti, v případě potřeby tak může při nastalých problémech poradit. Z výše uvedených důvodů bylo rozhodnuto respektovat předchozí výběr platební brány, použita tedy bude platební brána Stripe.

Platební brána Stripe

Stripe je platební brána, jíž využívají miliony podnikatelů k platebním transakcím. Poskytuje širokou škálu platebních produktů, jako jsou online platby, podpora předplatného, spravovat příjmy nebo zasílat faktury. Mimo to poskytuje snadné API pro rychlou integraci, a to včetně knihoven pro velké množství jazyků a platforem. Mezi nimi nechybí knihovna pro JavaScript, která bude využita při implementaci. [48]

Níže budou popsány pro nástroj Timer2Ticket klíčové věci, které platební brána nabízí.

API

Pro snadnou a nenáročnou integraci poskytuje platební brána Stripe velmi obsáhlé REST API. Toto API využívá pro obsah těl požadavků JSON objekty a při odpovědi používá standardní kódy HTTP. Pro autorizaci je využit API klíč, který lze získat ve Stripe dashboardu. Ten se v závislosti na výběru způsobu komunikace s API vloží do úvodní konfigurace. [49]

Obsáhlost API dokládá podpora veškerých funkcionalit popsaných dále v textu.

Webhooky

Stripe umožňuje naslouchat událostem, které se v systému platební brány dějí, a v nástroji, do kterého je Stripe integrován, na ně reagovat. Využitelné jsou především u asynchronních událostech, jako je například platba platební kartou nebo opakovaná platba předplatného.

Při vytváření webhooku ve Stripe dashboardu je nutné zadat URL endpointu, kde aplikace bude naslouchat a následně události, kterým bude naslouchán. Těch Stripe nabízí několik desítek. Na zadaný endpoint pak budou odesílány HTTP POST požadavky. Na ně je potřeba vrátit úspěšný stavový kód (`2xx`). Pro zabránění vypršení časového limitu je potřeba odpovědět ideálně ještě před tím, než se začne webhook zpracovávat. Pokud časový limit vyprší, Stripe po čase z bezpečnostních důvodů nastavený koncový bod deaktivuje.

Z důvodu bezpečnosti je vhodné zasílané webhooky zkontrolovat, zdali je původcem skutečně platební brána a nikoliv potenciální útočník. Stripe přidává do každého zaslaného webhooku podpis události. Ten je obsažen v HTTP hlavičce `stripe-signature`. Pro ověření podpisu a tedy ověření, že událost pochází z platební brány, je vhodné využít některou z oficiálních knihoven. Podpis je sice možné ověřit manuálně, avšak při implementaci lze předpokládat využití poskytnutých knihoven.

Pro ověření je nutné získat Signing secret z nastavení webhooků ve Stripe dashboardu. Tento secret je unikátní pro každý koncový bod. Následně stačí zavolat dle použité knihovny příslušnou knihovní funkci. Ta přijímá jako parametry původní (nijak nemodifikované) tělo zprávy, podpis zaslaný v HTTP hlavičce a právě secret koncového bodu. [50]

Testování

Pro snadnou integraci a především její vyzkoušení nabízí Stripe testovací režim. Ten se až na výjimky neliší oproti běžnému provozu platební brány. Testovací režim tak věrně simuluje veškerou platební infrastrukturu. Mezi daty z testovacího režimu a běžného provozu lze přepínat v dashboardu. Tento přepínač však neovlivňuje, jaký mód je použit, jelikož oba režimy mají vlastní API klíče.

V testovacím režimu nejsou účtovány poplatky za využívání platební brány, ani nejsou při transakcích prováděny peněžní operace. Pro snadné používání nabízí Stripe sadu testovacích karet, díky kterým lze simulovat různé scénáře. K dispozici jsou platební karty dle poskytovatele či země původu. Dále lze simulovat odmítnutí platby či zadání nesprávných údajů. Taktéž lze testovat systém prevence podvodů společnosti Stripe, zadání neplatných dat, sporné transakce či refundaci.

Další způsob testování jsou tzv. testovací hodiny. Těmi lze v testovacím režimu simulovat pohyb v čase dopředu. Tímto lze simulovat různé události, jako například opakovanou platbu předplatného. Testovací hodiny lze vytvořit a ovládat ve Stripe dashboardu.

Stripe umožňuje testovat i webhooky. Zasílané události lze snadno vyzkoušet při lokálním běhu aplikace. To je umožněno pomocí Stripe CLI, testovacího nástroje pro příkazový řádek, umožňujícího přesměrovávat události na příslušný `localhost` endpoint. Nejprve je však nutné tento nástroj nakonfigurovat. Stripe CLI je však možné využít pouze v testovacím režimu. [50]

Uživatelé

V platební bráně Stripe lze vytvářet uživatele. U nich lze evidovat nejen osobní informace, jako jméno, e-mailovou adresu nebo fakturační údaje, ale také dodací adresu, jazyk pro lokalizaci zobrazovaných webových stránek či přidání vlastních metadat. Mimo to je možné u zákazníka evidovat daňové údaje, jako je DIČ nebo daňová lokalita.

Nejpodstatnější jsou však funkcionality spojené s finančními transakcemi. U uživatele je možné evidovat předplatné či jednorázové platby. Uživatel si dále může uložit své platební údaje. Stripe také podporuje možnost zobrazovat u proběhlých plateb zákazníkovy faktury. [50]

Vytváření produktů

Základem pro platební bránu Stripe jsou produkty, respektive jejich ceny. Produkt odpovídá nabízené službě, kterou si zákazník bude moci zakoupit. K produktu je možné přiřadit několik cen, v závislosti na požadovaném cenovém modelu, avšak vždy je jedna z cen zvolena jako výchozí.

Při tvorbě produktu lze stanovit mnoho atributů, kromě popisu nechybí možnost nahrání obrázku či popis produktu. Stripe též poskytuje správu produktů, lze je tedy upravovat či mazat, respektive archivovat, pokud již byl produkt použit. To stejné lze říci i o vytvořených cenách produktu. [50]

Tvorba cen

Při vytváření ceny produktu je nutné zvolit cenový model, zdali je daň součástí ceny, samotnou cenu a zdali se jedná o jednorázovou nebo opakující se platbu. V případě opakující se ceny je dále nutné zvolit účtovací periodu.

Stripe nabízí celkem pět cenových modelů. V základním, „Standard pricing“, bude účtována pevně stanovená cena za každou zakoupenou jednotku. „Package pricing“ se liší tím, že stanovuje cenu ne za jednu, ale více jednotek. Cena se standardně zaokrouhluje nahoru, je-li tedy stanovená cena například 20 Kč za 10 jednotek, za 15 jednotek uživatel zaplatí 40 Kč.

„Graduated pricing“, neboli odstupňovaná cena, umožňuje stanovit fixní cenu a následně stanovit odlišnou cenu na základě počtu zvolených jednotek. To umožňuje například za prvních 5 jednotek stanovit fixní cenu na 100 Kč a 20 Kč za každou jednotku a poté již jen 10 Kč za další jednotku. Jsou tedy stanoveny vrstvy, při kterých se snižuje účtovaná částka za zakoupenou jednotku v dané vrstvě. Cena za jednotku se tedy liší na základě vrstvy dle počtu jednotek. Tímto modelem tak lze například zvýhodnit ty zákazníky, kteří kupují více jednotek produktů.

Předposlední cenový model je „Volume pricing“, tedy cena podle objemu. Tento model taktéž zvýhodňuje zákazníky kupující více jednotek. Oproti předchozímu modelu se liší tím, že cena za jednotku je stanovena na základě počtu zakoupených jednotek a je pro všechny jednotky shodná. Je-li cena stanovena na 10 Kč při koupi až pěti produktů a dále 5 Kč za produkt, při koupi 6 produktů tak zaplatí 30 Kč (6 produktů $\times$ 5 Kč za produkt) oproti „Graduated pricing“, kde by bylo účtováno 55 Kč (5 produktů za 10 Kč za produkt + 1 produkt za 5 Kč za produkt).

V posledním modelu je zákazníkovi umožněno stanovit si částku, kterou zaplatí. Je možné předzvolit doporučenou částku nebo zvolit minimální a maximální částku. Tento cenový model však nepodporuje opakované platby. [50, 51]

Při vytváření ceny produktu je možné podporovat více měn. Po přidání další měny Stripe navrhne aktuální směnný kurz, avšak je možné zvolit vlastní cenu. Při platbě pak Stripe určí měnu zákazníka dle jeho IP adresy, má-li zakoupený produkt v této měně. V opačném případě se použije výchozí měna. Výchozí měnou je ta měna, která je vybrána jako první při vytváření ceny produktu.

Pokud je použito v rámci jedné relace více produktů, musí všechny produkty podporovat měnu uživatele. Není-li tomu tak, je použita výchozí měna. Je tedy nutné, aby byla výchozí měna u všech produktů stejná. Není-li cena stejná, Stripe vrátí chybu. [50]

Opakované platby (předplatné)

Při opakovaných platbách je nutné vyřešit, jaké bude chování při změně, respektive zrušení předplatného. Změnou předplatného se rozumí zvolení jiného členství nebo množství okamžitých synchronizací. Snížením členství platební brána Stripe rozumí snížení původně účtované ceny, v opačném případě se jedná o zvýšení členství.

Při změně členství je možné nastavit, zdali se má cenový rozdíl zaplatit okamžitě po provedení změny, nebo až na konci fakturačního cyklu. V obou případech je vypočítána poměrná část, při okamžité úhradě je ihned vygenerována faktura a předplatné bude změněno pouze v případě úspěšné platby.

Vznikne-li při snížení členství přeplatek, například pří přechodu z placeného na neplacené členství, budou o tento přeplatek sníženy částky následujících faktur. Stripe však patrně neumožňuje vrátit tento zůstatek uživatele, v dashboardu je však možné manuálně upravit zůstatek a zrušit či vrátit část provedené transakce, touto formou by tedy případně bylo možné vrátit uživatelův zůstatek.

Stripe též nabízí velké možnosti chování v případě neúspěšného zaplacení opakované platby. Zrušení též může iniciovat zákazník, kdy je možnost nastavení, zdali má zrušení proběhnout okamžitě, nebo až na konci fakturačního cyklu. [50]

Kupóny

Platební brána Stripe podporuje kupóny, respektive propagační kódy. Uplatněním kupónu lze získat slevu na platbu, ať už procentuální nebo na stanovenou částku. Při vytváření kupónu ve Stripe dashboardu lze nakonfigurovat velké množství atributů, jako je výše slevy, výběr konkrétních produktů, na které může být kupón uplatněn, jak dlouho bude kupón platit po uplatnění, kolikrát lze kupón uplatnit nebo jeho platnost. Také lze vytvořit kód pro zákazníky, kde lze stanovit platnost pouze na první objednávku, omezení na konkrétního zákazníka, počet použití kupónu nebo minimální hodnotu objednávky.

Kupóny neumožňují stanovení produktů, které by byly zákazníkovi zdarma poskytnuty, což rozporuje navrhovaný monetizační model vytvořený v bakalářské práci Bc. Jakubem Čermákem [15]. Možným řešením je místo počtu okamžitých synchronizací prostřednictvím kupónu určit slevu na produkty, případně zachovat původně navrhovaný model a kupóny řešit vlastní implementací, nikoliv prostřednictvím platební brány. [50]

Zákaznický portál

Pro správu předplatného, změnu platební metody nebo zobrazení historii objednávek nabízí Stripe zákaznický portál. Zde mohou zákazníci jednoduše a efektivně provádět zmíněné akce. Zákaznický portál lze vytvořit ve Stripe dashboardu, tedy bez nutnosti cokoliv programovat a poskytuje velké množství přizpůsobení. Lze zvolit, co bude zobrazeno a jaké operace bude moci uživatel vykonávat, stejně tak jako změna vzhledu formou vložení loga, nadpisu nebo výběrem barev. Zákaznický portál však má i několik omezení, neumožňují například aktualizovat předplatné s více produkty. [50, 52]

Daně

Stripe umožňuje automaticky vypočítat a následně vybrat daň z provedené platby. Platební brána poskytuje tzv. Stripe Tax, který se snaží řešit daňová pravidla na straně platební brány a nezatěžovat tak podnikatele využívající tuto platební bránu složitou legislativou v oblasti daní.

Tato platforma má provádět automatický výběr daní pomocí jednoduché konfigurace. Na základě země původu zákazníka vypočítá a vybere správnou výši daně. Taktéž umí u evropských a australských zákazníků ověřit číslo identifikující daňový subjekt, v České republice DIČ. Stripe Tax následně umožňuje exportovat příslušné reporty pro odvod daní. [53]

Nastavení daní lze provést ve Stripe dashboardu zadáním země, ve kterém je provozováno podnikání, výběru kategorie produktu a zvolení země registrace u daňového úřadu. Na základě těchto údajů spolu s údajem země původu zákazníka jsou příslušné daně počítány. V nastavení je poté ještě možné zvolit, zdali je daň již zahrnuta v ceně, případě má-li být cena produktu navýšena o daň. [50]

Evropská unie určuje pravidla stanovení DPH dle typu služby. Stripe tyto pravidla uplatňuje. Výpočet daně je však velice komplexní proces závisející na mnoha faktorech. Mimo tipu služby závisí také na zemi kupujícího (zdali je nebo není v EU), zdali je zákazník podnikající či nepodnikající osoba nebo zdali je dodavatel plátce DPH, neplátce DPH nebo identifikovaná osoba. DPH závisí na potenciálním osvobození plnění od daně. Na základě výše uvedeného se stanoví daň, kdy může být použita česká DPH, přenesená DPH nebo bez DPH. Detailněji je daná problematika popsána v citovaných zdrojích. [54, 55, 56]

2.3 Sběr požadavků od zadavatele

Vzhledem k tomu, že nástroj Timer2Ticket zadavatel využívá i pro svou vlastní potřebu, je nutné též podrobit analýze jeho požadavky, kterými by měl Timer2Ticket disponovat. Jejich sběr probíhal formou diskuzí během pravidelných konzultací této diplomové práce.

2.3.1 Integrace nástroje Jira

V této části textu bude analyzováno, zdali má smysl Jiru do nástroje Timer2Ticket integrovat. Jiru lze považovat za jeden z nejrozšířenějších projektových nástrojů, proto je překvapením, že tento nástroj není zatím podporován. Bude tedy provedeno i porovnání Jiry s nástrojem Redmine.

Zatímco Redmine je open-source platforma, která je k dispozici zdarma, Jira je komerční nástroj vyvinutý společností Atlassian. Obsahuje několik placených verzí, ale i omezenou verzi zdarma. Obecně lze však říci, že nabízí v mnohém pokročilejší funkce. Jira poskytuje možnost plánování projektů, tvorbu agilních boardů a roadmap, sledování chyb, řízení verzí nebo reportování. Také poskytuje velké možnosti šablon pro vývoj softwaru a přizpůsobení kupříkladu životního cyklu požadavku. Na první pohled viditelný rozdíl může být grafické rozhraní, které má Jira modernější a přehlednější. [7, 57, 58]

Popularitu Jiry dokládá její využití ve více než 100 000 organizací [7]. Její oblíbenost dokazuje i podíl na trhu, který dosahuje dle [59] přes 41 %. Pro porovnání podle stejného zdroje dosáhl Redmine 0,58 %. Dle výše uvedeného má patrně smysl Jiru zaintegrovat do vyvíjeného nástroje Timer2Ticket.

Jira API

Integrace Jiry by probíhala, podobně jako Redmine, prostřednictvím REST API, které i tento nástroj vystavuje. Současné API je verze 3, které je však v betaverzi. Je totožné s API verze 2, avšak navíc poskytuje podporu pro *Atlassian Document Format* v těle komentářů, komentářů pracovních záznamů, v popisu požadavků atd. Tyto nové funkce se však ještě mohou měnit. [60]

Samotné API je velmi rozsáhlé, z tohoto důvodu budu popisovat pouze relevantní informace pro nástroj Timer2Ticket. Pro integraci Jiry do Timer2Ticket je podstatná práce přes API s projekty, požadavky a časovými záznamy. Jira REST API umožňuje u těchto entit vytváření, čtení, úpravu i mazání, tedy vše potřebné.

Autentizace a autorizace

Pro zajištění bezpečnosti a ochrany dat před neoprávněným přístupem v Jira je zapotřebí požadavky volané přes REST API autorizovat. Předtím je však nejprve potřeba vygenerovat API token. Ten lze vygenerovat ve správě účtu v sekci Security, kde po kliknutí na „Create and manage API tokens“ vytvářet nové tokeny nebo zrušit již vytvořené.

Po vytvoření tokenu můžeme požadavky pomocí vygenerovaného API tokenu a uživatelského jména autorizovat. Nejprve je třeba připravit řetězec ve formátu `user_mail:api_token`, který je následně zakódován do Base64. Zakódovaný řetězec je následně nutné vložit do hlavičky Authorization. [61, 60]

Stránkování

Pro zvýšení výkonu je v Jira REST API využito stránkování. Děje se tak pro operace obvykle vracející velké množství položek. V těchto případech jsou odpovědi obaleny JSON objektem obsahujícím stránkovací metadata. Těmito metadaty jsou `startAt`, `maxResults`, `total` a `isLast`.

Celkový počet položek obsažený ve všech stránkách je obsažen v `total`. Tato hodnota se však v průběhu dotazování může měnit, proto je třeba brát v úvahu možnost vrácení prázdné stránky.

Maximální počet položek obsažených na jedné stránce se nachází v `maxResults`. Tato hodnota se liší pro různé operace. Tento počet lze zadat do *query string* části URL dotazu. V případě, že je v dotazu stanovený `maxResults` větší, než umožňuje API, je tato hodnota ignorována a využije se výchozí maximální hodnota daného dotazu.

S předchozím odstavcem silně souvisí `startAt` značící index první vrácené položky na dané stránce. Stejně jako *query string*, i tato metadata lze vložit do *query string* části při dotazování.

Zbývající hodnotou je pak `isLast`. Tento příznak je typu *boolean* a značí, že vrácená stránka je stránkou poslední daného dotazu. Je však třeba mít na paměti, že tato metadata nemusí být vrácena v každém požadavku. [60]

Filtrování a řazení

Jira REST API poskytuje i široké možnosti v oblasti filtrování a řazení, kterých lze dosáhnout pomocí Jira Query Language (JQL). Jedná se o dotazovací jazyk, který poskytuje možnost vytvářet složitější dotazy v nástroji Jira, včetně potřebného filtrování a řazení. Díky JQL je možné vyfiltrovat například požadavky dle stavu a poté je seřadit pomocí data modifikace, což umožní zefektivnit synchronizaci objektů. JQL je patrně srozumitelný a snadno naučitelný, je velmi podobný jazyku SQL. Taktéž připomíná jednoduché anglické věty. [62]

Vyhledávat pomocí JQL je možné nejen v grafickém rozhraní aplikace, ale taktéž pomocí API. Zde stačí vložit vyhledávací výraz do parametru `jql` v *query string* části URL dotazu. [60]

Webhooky

Jira REST API podporuje i webhooky. Detailněji se tomuto tématu věnuje sekce Webhooky v Jira.

Synchronizace mezi projektovými nástroji

Na jedné z konzultací s vedoucím práce bylo řečeno, že společnost Jagu s.r.o. potřebuje synchronizovat časové záznamy mezi Jirou a Redmine. Pro více informací byl požádán projektový manažer společnosti Ing. Oldřich Malec a Bc. Jakub Čermák, tvůrce nového frontnedu o konzultaci. Ti sdělili více informací a detailů ohledně plánované synchronizace. Z konzultace vzešly dva

požadavky na způsob synchronizace časových záznamů mezi těmito systémy, respektive mezi projektovými nástroji obecně. Konzultací se mimo jiné potvrdila analýza nástroje Jira a smysluplnost jeho integrace do Timer2Ticket.

Prvním požadavkem je synchronizace časových záznamů podobně, jako to dělá Timer2Ticket nyní. Synchronizovaly by se tedy všechny časové záznamy uživatele ze všech jeho projektů. Představa je taková, že by požadavek v jednom projektovém nástroji úkolu obsahoval ve vlastním atributu odkaz na ekvivalentní požadavky v druhém nástroji. Na základě tohoto atributu by pak byly časové záznamy synchronizovány. V případě, že by požadavek odkaz neměl, časové záznamy z tohoto úkolu by se synchronizovávaly do předem definovaného výchozího požadavku pro požadavky bez odkazu.

Druhý požadavek je pak synchronizace časových záznamů celého projektu, tedy všech uživatelů, kteří vkládají časové záznamy. To by bylo možné realizovat vytvořením speciálního účtu s přístupem do celého projektu v obou projektových nástrojích, přes který by synchronizace probíhala. Zde ale bude třeba vytvořit mapování uživatelů z jednoho systému do druhého, aby bylo možné synchronizovaný časový záznam přiřadit správnému účtu ve druhém projektovém nástroji. Pokud by však spojení neexistovalo, podobně jako u požadavků bez odkazu by zde existoval předem definovaný výchozí uživatel, kterému by byly přiřazeny časové záznamy, pokud by mapování daného uživatele chybělo. Samotná synchronizace časových záznamů pak bude probíhat stejně jako v prvním požadavku, u požadavku tedy bude ve vlastním atributu odkaz na požadavek, do kterého se budou časové záznamy synchronizovat. Nebude-li odkaz existovat, budou časové záznamy synchronizovány do výchozího požadavku, který bude stejně tak definován při konfiguraci spojení.

Z výše uvedeného vyplývá, že není zájem o synchronizaci konfiguračních objektů. To by značně zvýšilo složitost implementace, jelikož by bylo pravděpodobně nutné synchronizovat i další atributy, jako například stav požadavku. Navíc, rozšíření nástroje Timer2Ticket o synchronizaci konfiguračních objektů by bylo mimo rozsah aplikace, jelikož jeho primární zaměření je synchronizace časových záznamů.

2.3.2 Převod faktur do nástroje Fakturoid

Vzhledem k tomu, že provozovatelem nástroje Timer2Ticket bude společnost Jagu s.r.o., která pro vedení účetnictví využívá nástroj Fakturoid, je vyžadováno evidování faktur právě v tomto nástroji. Fakturoid umožňuje kupříkladu vystavování faktur, párování plateb s bankou, exporty do účetních systémů nebo evidenci nákladů. [63]

Podstatné pro vývojáře nástroje Timer2Ticket je však možnost integrace pomocí vystaveného REST API. Mimo to taktéž poskytuje dvě oficiální knihovny a několik komunitních, nezahrnuje však verzi pro JavaScript. Pro snadný vývoj je možné požádat provozovatele o bezplatný vývojářský účet. [64]

API je popsáno v oficiální dokumentaci. Podstatná je práce s fakturami, kdy API umožňuje vytvoření, úpravu a smazání faktury a následně její stažení ve formátu PDF. Následně lze pomocí API faktury získat, při dotazování je možné aplikovat velké množství filtrů. [65]

2.3.3 Zpětná kompatibilita

Další validní připomínkou zadavatele je zpětná kompatibilita s původní verzí Timer2Ticket. Z diskuze vzešlo, že nebude možné zachovat původní databázové schéma, mimo jiné i z důvodu podpory více spojení uživatele, a tak budou stávající data nástroje převedena na novou verzi aplikace.

Data uživatelského účtu původní verze aplikace budou v co nejvyšší míře zachována. V praxi to znamená, že přihlásí-li se uživatel v nové verzi nástroje se svým starým e-mailem, uvidí svá původní data. Měl-li uživatel v původní verzi vytvořené spojení, bude mu vytvořeno i v nové verzi.

2.4 Požadavky

Na základě předcházejícího textu věnující se analýze současného stavu nástroje a jeho možného budoucího směřování a sběru požadavků od zadavatele vzešly následující požadavky na budoucí rozvoj nástroje Timer2Ticket. Požadavky jsou pro přehlednost členěny do logických celků a každý obsahuje prioritu, odhad náročnosti, výčet dotčených částí aplikace a typ požadavku. Priority jednotlivých požadavků byly konzultovány a následně schváleny zadavatelem této práce.

Odhad pracnosti byl rozdělen do tří úrovní – nízká, střední a vysoká. Nízká úroveň pracnosti zahrnuje požadavky, které jsou snadno a rychle splnitelné a které nevyžadují velké úsilí k dokončení. Střední úroveň pracnosti je o něco náročnější, než předchozí úroveň. Tyto požadavky jsou již složitější a vyžadují již hlubší znalost aplikace. Při implementaci daného požadavku může být vyžadován zásah do více než jedné komponenty. Poslední, vysoká úroveň pracnosti vyžaduje hlubokou znalost všech částí zdrojových kódů aplikace, jelikož je potřeba požadavky implementovat skrz všechny části aplikace, popřípadě je požadavek časově velmi náročný.

Pro stanovení priority byla využita metodika MoSCoW. Cílem metodiky je usnadnění rozhodování, které požadavky budou splněny, dojde-li k omezení zdrojů. Prioritizací by tak nemělo dojít k odkladu realizace pro vyvíjený systém klíčových částí, pokud se vývojový tým dostane například pod časový nebo jiný tlak. Tento populární přístup je využíván při správě požadavků, které kategorizuje do čtyř skupin: [66]

Must have – musí být Požadavky spadající do této kategorie jsou nejvyšší prioritou a nezbytné pro správnou funkčnost aplikace. Jedná se tedy o klíčové prvky pro splnění hlavních cílů při vývoji aplikace a bez jejich implementace nebude aplikace funkční.

Should have – mělo by být Požadavky z této kategorie jsou pro aplikaci důležité, avšak nikoliv kritické pro základní funkčnost. Nerealizování těchto požadavků tak neohrozí chod aplikace, avšak její rozšíření o tento typ požadavků výrazně zvýší hodnotu aplikace.

Could have – může být Tyto požadavky nejsou nezbytné pro základní funkci aplikace. Oproti předchozí kategorii jsou požadavky z této kategorie méně důležité pro finální verzi produktu a obvykle se právě tyto požadavky nerealizují, nejsou-li dostupné zdroje.

Won't have – nemá být Tato skupina požadavků nebude v aktuální fázi vývoje realizována. Požadavky však mohou být realizovány v některé z budoucích verzí aplikace. [66]

Typ požadavků byl stanoven pomocí metodiky FURPS. Tento akronym v sobě skrývá pět pojmů pro kategorizaci požadavků. Prvním z nich je functionality (funkčnost) zahrnující funkční požadavky. Jedná se o specifikaci, co zákazník vyžaduje a jakým způsobem to bude realizováno. Zbývající oblasti se pak zabývají nefunkčními požadavky.

Usability (použitelnost) je zaměřena na uživatelské rozhraní zahrnující například snadné používání, intuitivnost nebo efektivitu. Reliability (spolehlivost) se týká spolehlivosti a stability systému. Může se jednat o odolnost proti chybám či schopnost zotavení se při poruše. Performance (výkon) se zabývá výkonem – rychlostí, výkonem nebo odezvou. Spadá sem též omezení zdrojů při běhu (například čas procesoru nebo operační paměť). Poslední oblastí je pak supportability (podporovatelnost), která se zaměřuje na snadný provoz, údržbu, podporu rozšiřitelnost systému. [67]

Při výběru metodiky byla dále zvažována metodika FURPS+, která rozšiřuje původní metodiku FURPS o další aspekty, jako jsou požadavky na design, implementaci, rozhraní nebo fyzické vlastnosti. Upřednostněn byl však FURPS, jelikož poskytuje kategorizaci, která se ve fázi analýzy pro tuto diplomovou práci zdála být dostačující. [68, 69]

Napojení nového frontendu na backend

Pro základní funkci nástroje Timer2Ticket je potřeba napojit nový frontend na backend, což je jedna z priorit této diplomové práce. To zahrnuje úpravy, rozšíření či doplnění všech částí aplikace.

P1 – Sjednocení autentizační a autorizační platformy

Backend a nový frontend budou využívat stejnou platformu pro autentizaci a autorizaci uživatelů, a to Auth0. Pro přihlašování bude uživatelům umožněno využívat i vybrané sociální sítě.

Náročnost: Střední

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P2 – Podpora více spojení jednoho uživatele

Každý uživatel bude moci vytvořit i více než jen jedno spojení. Nový frontend to již podporuje, stávající backend předpokládá pouze jedno spojení na uživatele.

Náročnost: Vysoká

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P3 – Správa spojení

Pod správu spojení patří možnost změny konfigurace nástrojů k synchronizaci, změny plánu synchronizace a změny aktivity spojení.

Změna konfigurace zahrnuje změnu nástrojů, mezi kterými budou časové záznamy synchronizovány, popřípadě změnu konfigurace daných nástrojů. Změnou plánu synchronizace se rozumí možnost změnit plán synchronizace konfiguračních nebo časových objektů (v případě komerční verze dle omezení jednotlivých členství). Změna aktivity spojení značí, zdali časové záznamy mezi těmito systémy budou synchronizovávány dle plánů synchronizace či nikoliv.

Náročnost: Vysoká

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P4 – Možnost okamžité synchronizace spojení

Aplikace bude umožňovat možnost okamžité synchronizace daného spojení. To bude vyvoláno stisknutím příslušného tlačítka v uživatelském rozhraní aplikace. Uživatel bude moci vyvolat samostatně buďto synchronizaci konfiguračních objektů nebo časových záznamů. V případě komerční verze bude uživateli umožněno provést okamžitou synchronizaci pouze v případě dostupných volných okamžitých synchronizací uživatele.

Náročnost: Vysoká

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P5 – Zobrazení logů

Uživatel si bude moci zobrazit stránku s logy jobů spojení. Ty budou zobrazovat v tabulce informaci o spojení, jemuž log daného jobu náleží, poté datum naplánování a dokončení, typ jobu, původce a status s případnou chybovou hláškou, nebude-li job úspěšně dokončen. Uživatel si bude moci výsledky vyfiltrovat dle každého sloupce.

Uživatel si bude moci zobrazit i přímo logy patřící danému spojení ze stránky spojení, a to kliknutím na příslušné tlačítko v boxu spojení, jehož logy si přeje zobrazit.

Náročnost: Vysoká

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P6 – Nastavení e-mailu

Jelikož aplikace bude podporovat přihlášení pomocí sociálních sítí, může se stát, že nebude možné získat z dané služby e-mailovou adresu uživatele (například sociální síť Facebook při registraci nevyžaduje zadání e-mailové adresy). Z tohoto důvodu uživatel bude moci zadat e-mail, nebude možné ho získat při registraci, respektive prvním přihlášení. E-mail však již nebude moci sám měnit. O této skutečnosti však bude informován. Bude-li si chtít e-mail změnit, bude nutné kontaktovat provozovatele nástroje Timer2Ticket.

Náročnost: Střední

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P7 – Správa účtu

Uživateli bude umožněno spravovat svůj účet. To zahrnuje nastavení zasílání notifikací na e-mail a možnost změny hesla, není-li k přihlášení uživatele používána sociální síť. Ve správě účtu však nebude zahrnuto smazání účtu, které bude možné realizovat kontaktováním provozovatele.

Náročnost: Střední

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P8 – Volba časové zóny

Uživatel si bude moci v nastavení zvolit časovou zónu, na základě které budou plánovány synchronizační joby. Po přihlášení bude uživateli zvolena výchozí časová zóna (bude se jednat o středoevropský čas, popřípadě středoevropský letní čas, bude li v danou chvíli platný). Časovou zónu si však uživatel bude moci kdykoliv měnit. Při změně časové zóny nebudou měněny zvolené časy synchronizace.

Náročnost: Střední

Priorita: Could have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P9 – Možnost volby notifikací

Uživatel si bude moci v nastavení vybrat, jaký typ notifikací bude dostávat na svůj e-mail, bude-li zadán. V současné chvíli bude na výběr pouze možnost informování v případě problému při běhu synchronizačního jobu.

Náročnost: Střední

Priorita: Could have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P10 – Zasílání notifikací

Uživateli budou zasílány e-maily s notifikacemi na základě konfigurace jejich zasílání.

Náročnost: Střední

Priorita: Could have

Dotčené části: Backend

Typ požadavku: Funkční – funkčnost

Zprovoznění monetizace

Monetizace se týká pouze komerční verze nástroje Timer2Ticket. Způsoby monetizace byly navrženy a naimplementovány v diplomové práci Bc. Jakuba Čermáka. Ten navrhl parametry jednotlivých nabízených produktů včetně cen. Implementace však proběhla pouze na straně frontendu. Bude tedy třeba nejen dokončit stávající frontend a rozšířit pro nové funkce podporu backendové části, ale i například propojit aplikaci s platební branou nebo řešit zobrazování faktur zákazníkům.

P11 – Výběr členství

Aby mohl uživatel využívat nástroj Timer2Ticket, bude si muset zvolit jedno z nabízených členství. Budou nabízeny tři členství (hobby, junior, senior) lišící se v počtu aktivních spojení, četnosti synchronizace a ceně. Hobby členství bude nabízeno zdarma.

Po prvním přihlášení se uživateli zobrazí tabulka popisující jednotlivá členství. Pro pokračování si musí jedno zvolit. Zvolí-li si zpoplatněné členství, bude přesměrován na stránku pro uhrazení měsíčního poplatku. Poplatek následně bude ze zadané platební karty strháván každý měsíc. Bude-li platba neúspěšná, současné členství se zruší, respektive se převede na nezpoplatněné hobby členství.

Své členství bude moci uživatel kdykoliv změnit. To proběhne opět zobrazením nabídky členství formou tabulky. Po zvolení vybraného členství bude opět v případě zpoplatněného členství přesměrován na stránku pro úhradu částky za členství. Nevyužije-li uživatel celé předplacené původní členství, bude mu poskytnuta sleva na novou úhradu. Nová částka k zaplacení poplatku tedy bude snížena o úměrnou část nevyužitého původního předplatného.

Při změně členství bude konfigurace spojení upravena tak, aby v případě snížení členství odpovídala parametrům nově zvoleného členství. Při zvýšení úrovně členství nebude nutné spojení jakkoliv upravovat.

Náročnost: Vysoká

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P12 – Přikoupení aktivních spojení nad rámec členství
Nebude-li uživateli vyhovovat počet aktivních spojení v daném členství, bude si moci za stanovenou částku přikoupit další. Aktivní spojení nad rámec členství bude možné pořídit s výběrem nebo změnou členství.

Náročnost: Vysoká

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P13 – Podpora okamžitých synchronizací
Uživatel bude moci provést okamžitou synchronizaci synchronizačních objektů nebo časových záznamů, bude-li mít dostatek okamžitých synchronizací. Ty však budou zpoplatněny. Okamžité synchronizace si bude možné pořídit jednorázovou platbou, kdy bude účtována určitá částka za balíček obsahující daný počet okamžitých synchronizací.

Náročnost: Vysoká

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P14 – Historie objednávek
Uživatel si bude moci zobrazit historii svých objednávek. Bude mu zobrazeno datum, uhrazená částka a zakoupené položky.

Náročnost: Střední

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P15 – Okamžité ukončení předplatného
Uživatel bude moci okamžitě ukončit stávající členství. To bude realizováno změnou členství na základní hobby členství, které je zdarma.

Náročnost: Střední

Priorita: Should have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P16 – Logování použitých okamžitých synchronizací
Použije-li uživatel okamžitou synchronizaci, bude zalogován čas a datum použití okamžité synchronizace, typ okamžité synchronizace a spojení. Uživatel si bude moci kdykoliv zobrazit seznam použitých okamžitých synchronizací.

Náročnost: Vysoká

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P17 – Napojení platební brány
Bude-li vybrán zpoplatněný produkt, uživatel bude přesměrován na platební bránu Stripe. Po uhrazení částky se provede změna na jeho Timer2Ticket účtu (tedy bude změněno členství nebo připsány okamžité synchronizace).

Náročnost: Vysoká

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P18 – Možnost stažení faktury
U každé objednávky si uživatel bude moci stáhnout fakturu.

Náročnost: Vysoká

Priorita: Should have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P19 – Převod faktur do nástroje Fakturoid
Veškeré uhrazené faktury budou převáděny do nástroje Fakturoid.

Náročnost: Vysoká

Priorita: Could have

Dotčené části: Backend

Typ požadavku: Funkční – funkčnost

P20 – Uložení platebních údajů
Uživatel si bude moci uložit platební údaje. Ty bude možné upravovat, popřípadě smazat.

Náročnost: Vysoká

Priorita: Should have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P21 – Podpora kupónů
Aplikace bude podporovat možnost zadání kupónu. Zadá-li uživatel platný kupón, bude mu připsán určitý počet okamžitých synchronizací. Počet okamžitých synchronizací bude stanoven pro každý kupón samostatně.

Náročnost: Vysoká

Priorita: Could have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

Oddělení komerční a nekomerční části

Aplikace bude zveřejněna jakožto open-source, zároveň však bude provozována i její komerční verze. S ohledem na licenci, pod kterou bude aplikace vydána, však není možné na poskytování nástroje Timer2Ticket vydělávat (vyjma zadavatele této práce). Z tohoto důvodu bude nutné oddělit komerční a nekomerční (open-source) verzi aplikace.

P22 – Oddělení částí zdrojových kódu

Dojde k oddělení částí zdrojových kódů aplikace, které jsou využity pro komerční verzi. To umožní zveřejnit zdrojové kódy jakožto open-source bez možnosti nasazení komerční verze aplikace jinou osobou, než je zadavatel práce. Musí však být zachována funkčnost obou částí aplikace, tedy jak nekomerční (bez částí zdrojového kódu pro komerční verzi aplikace), tak i komerční verze.

Oddělení částí zdrojových kódů bude realizováno tak, že co největší možné části budou obsaženy v separátní knihovně, která nebude zveřejněna, popřípadě tyto části budou deaktivovány nebo nebudou zobrazovány v grafickém rozhraní aplikace. Knihovna bude poskytovat metody, funkce a objekty, které budou moci být volány ze zveřejněné open-source verze, bude-li to v konfiguraci aplikace povoleno. Konfigurací tak bude zajištěno, že knihovna nebude použita, nebude-li ji mít uživatel k dispozici.

Náročnost: Vysoká

Priorita: Must have

Dotčené části: Backend, frontend

Typ požadavku: Nefunkční – podporovatelnost

Přidání nástroje Jira

Do budoucna se předpokládá s přidáním dalších nástrojů k synchronizaci. Prvním z nich bude projektový nástroj Jira.

P23 – Přidání nástroje Jira

Timer2Ticket bude podporovat synchronizaci časových nástrojů s nástrojem Jira. To znamená, že bude umožněno synchronizovat časové nástroje mezi Jirou a stávajícími podporovanými nástroji k synchronizaci, tedy s Toggl Track a Redmine.

Náročnost: Vysoká

Priorita: Should have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P24 – Podpora synchronizace časových záznamů mezi projektovými nástroji

Mezi projektovými nástroji se nebudou synchronizovat konfigurační objekty. Bude zde pouze mapování těchto objektů z jednoho projektového nástroje do druhého, na základě kterého se budou synchronizovat časové záznamy. Budou synchronizovány všechny časové záznamy uživatele (ze všech projektů).

Mapování bude vytvořeno na základě vlastního atributu, který stanoví uživatel při konfiguraci. Nebude-li existovat mapování, bude zde výchozí objekt, do kterého budou časové záznamy synchronizovány. Výchozí objekt bude taktéž stanoven uživatelem při konfiguraci spojení. Z tohoto důvodu bude pro jedno spojení pouze jednosměrná synchronizace, jeden projektový nástroj tedy lze považovat za primární. Bude-li potřeba vytvořit obousměrnou synchronizaci, lze tak učinit vytvořením dvou spojení.

Na základě tohoto mapování pak budou synchronizovávány jednotlivé časové záznamy. Bude-li existovat mapování daného objektu v primárním nástroji, kterému náleží časový záznam, bude vytvořen časový záznam ve druhém nástroji. V případě smazání nebo změny mapování se již namapované časové záznamy přenesou do nově zvoleného objektu, respektive do výchozího objektu.

Náročnost: Vysoká

Priorita: Should have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

P25 – Přidání „enterprise“ verze synchronizace mezi projektovými nástroji

„Enterprise“ verze synchronizace časových záznamů mezi projektovými nástroji zahrnuje synchronizaci všech časových záznamů jednoho projektu, synchronizovány tedy budou časové záznamy všech uživatelů. Z tohoto důvodu bude vyžadován přístup do synchronizovaných projektů v obou nástrojích. Nebudou však zahrnuty podprojekty, které podporuje například Redmine.

Logika bude stejná jako v případě synchronizace časových záznamů mezi projektovými nástroji pro jednoho uživatele. V případě „enterprise“ verze však bude třeba navíc řešit mapování uživatelů, kdy bude existovat mapování pro každého uživatele z jednoho systému do druhého. V případě neexistence mapování bude existovat výchozí uživatel, kterému budou synchronizované časové záznamy přiřazeny. V případě změny mapování uživatelů bude chování stejné jako při změně mapování konfiguračních objektů.

Při konfiguraci spojení tedy bude muset uživatel zadat kromě informací o připojení k dané službě stejně tak atribut, který obsahuje odkaz do druhého projektového nástroje a výchozí úkol, do kterého se budou synchronizovat časové záznamy v případě neexistence mapování. V případě synchronizace času celého projektu pak zde ještě bude správa mapování uživatelů a výchozí uživatel, kterému budou přiřazeny časové záznamy uživatelů s neexistujícím mapováním.

Náročnost: Vysoká

Priorita: Should have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

Zpětná kompatibilita

Jelikož je rozšiřována již existující produkční aplikace, bude třeba zajistit zpětnou kompatibilitu s původní verzí tak, aby uživatelé mohli plynule přejít na novou verzi.

P26 – Migrace původní verze Timer2Ticket na novou verzi

Všem uživatelům původní verze nástroje budou převedena již nakonfigurovaná spojení do současné verze, včetně mapovacích objektů. Uživatel se ke zmigrovanému spojení dostane pomocí registrace se stejnou e-mailovou adresou, která byla použita v původní verzi aplikace.

Náročnost: Vysoká

Priorita: Must have

Dotčené části: Backend

Typ požadavku: Nefunkční – podporovatelnost

Údržba aplikace

Potřebná je též údržba aplikace, která zajistí správné fungování a výkon aplikace po dobu chodu aplikace. Údržba mimo jiné může zahrnovat například bezpečnostní aktualizace, povýšení verzí používaných API či knihoven nebo opravu chyb.

P27 – Povýšení použitých knihoven na novější verze
Bude-li to nutné, budou povýšeny použité knihovny na takovou verzi, aby neobsahovaly chyby nebo zranitelnosti, popřípadě poskytovaly nové funkce umožňující vylepšení aplikace či zlepšení kompatibility, eventuálně podpořily snazší údržbu aplikace.

Náročnost: Střední

Priorita: Could have

Dotčené části: Backend, frontend

Typ požadavku: Nefunkční – podporovatelnost

P28 – Povýšení Toggl Track API
Pro komunikaci s nástrojem Toggl Track API bude použito nejnovější verze API, v době práce na této diplomové práci se jedná o Toggl Track API verze 9 a Reports API verze 3.

Náročnost: Střední

Priorita: Must have

Dotčené části: Backend

Typ požadavku: Nefunkční – podporovatelnost

Zvýšení efektivity

V rámci úprav a vylepšení nástroje bude nutné zapracovat na zvýšení efektivity Timer2Ticket, a to jak po výkonové stránce, tak i v případě potenciálních problémů umožňující například obejít monetizační model aplikace.

P29 – Prevence obcházení okamžitých synchronizací
Při změně konfigurace nástroje se okamžitě provede první synchronizace. Tímto tedy lze provést okamžitou synchronizaci. Stejně tak změnou plánu synchronizace lze vyvolat synchronizaci v blízké době.

Náročnost: Střední

Priorita: Could have

Dotčené části: Backend, frontend

Typ požadavku: Nefunkční – spolehlivost

P30 – Podpora webhooků
Pro synchronizaci konfiguračních objektů a časových záznamů bude využito webhooků, je-li to umožněno synchronizovaným nástrojem. Jejich implementace bude promítnuta do nabídky členství, která by se mohla rozšířit například o možnost okamžité synchronizace.

Náročnost: Vysoká

Priorita: Won't have

Dotčené části: Backend, frontend

Typ požadavku: Funkční – funkčnost

Dokončení nového frontendu

Bc. Jakub Čermák, autor nového frontendu, vydefinoval po dokončení své bakalářské práce velký počet úkolů upravující jeho implementaci. Tyto úkoly vzešly z testování použitelnosti webové části nástroje Timer2Ticket. V budoucnu by je tedy bylo vhodné implementovat.

Úkoly byly převzaty právě z bakalářské práce Bc. Jakuba Čermáka [15], respektive z vytvořených úkolů v nástroji Redmine tohoto projektu.

P31 – Přidání notifikací

Provede-li uživatel akci, bude mu zobrazena notifikace. V případě úspěšného dokončení požadované akce ji notifikace potvrdí, bude-li akce neúspěšná, zobrazí chybovou hlášku.

Zobrazení notifikací při neúspěchu akce implementováno není, potvrzení akce chybí u uložení změny plánu synchronizace či změny aktivity spojení v sekci spojení a při změně notifikace nebo časového pásma v sekci nastavení profilu.

Náročnost: Nízká

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P32 – Zobrazení probíhající okamžité synchronizace

Bude-li u spojení probíhat okamžitá synchronizace, uživateli to bude znázorněno formou načítacího kolečka na daném tlačítku, respektive textem v záhlaví boxu.

Náročnost: Nízká

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P33 – Změny barvy a aktivity tlačítek, když uživatel může vykonat akci

Některým tlačítkům zůstane šedé podbarvení, i když jsou aktivní. Jedná se o tlačítka se změnou hesla, použitím kuponu a změnou platebních údajů.

Náročnost: Nízká

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P34 – Odstranění možnosti notifikací o obdržených okamžitých synchronizacích

Jelikož bylo zjištěno, že není zájem o zasílání notifikací o obdržených okamžitých synchronizacích, tato možnost bude z aplikace odstraněna.

Náročnost: Nízká

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P35 – Revize responsibilního zobrazení

Při změně velikosti obrazovky se aplikace nechová správně – rozhodí se zarovnání prvků či se prvky nesprávně zobrazují. Taktéž bude změněno rozložení prvků boxu spojení na mobilních zařízeních (respektive zařízeních s úzkým displejem) tak, aby informace byly zobrazeny po řádcích.

Náročnost: Nízká

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P36 – Úprava uživatelského rozhraní

Nový frontend nabízí na mnohých místech možnost vylepšení, která zvýší uživatelský komfort při používání aplikace. Aplikace umožní potvrdit zadání kuponu stiskem klávesy enter a formátovat čas dle prohlížeče.

Náročnost: Nízká

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P37 – Úprava boční lišty

Bude umožněno zmenšení boční lišty. Obsaženo zde bude tlačítko na její otevření, respektive skrytí.

Náročnost: Nízká

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P38 – Úprava boxů spojení

Bude přepracován box spojení. Tlačítko pro změny aktivity bude při scrollování zafixováno a text u tohoto tlačítka bude změněn na aktivní, respektive neaktivní dle současného stavu spojení. Boxům bude změněna barva tak, aby nesplývaly, a k tlačítkům budou přidány ikony.

Dále bude možné boxy spojení seřadit dle potřeby uživatele. Boxy bude možné pojmenovávat. Při synchronizaci každou hodinu nebude zobrazován čas synchronizace, který je pro tuto volbu irelevantní.

Náročnost: Střední

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P39 – Úprava formuláře s platebními údaji

Ve formuláři s platebními údaji budou vyznačena povinná pole spolu s nápovědou, je-li potřeba. Pole pro DIČ bude validováno pomocí regulárního výrazu. Země se bude vybírat pomocí rozbalovací nabídky zemí.

Náročnost: Střední

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P40 – Volba členství kliknutím na sloupec

Uživatel si bude moci vybrat členství kliknutím na celý sloupec v tabulce s nabídkou členství, nikoliv jen na tlačítko, jako je tomu v nynější verzi.

Náročnost: Nízká

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P41 – Předvýběr konkrétního členství při vstupu z propagačního webu

Přejde-li uživatel do aplikace z propagačního webu kliknutím na vybrané členství, toto členství již bude zvoleno při načtení stránky.

Náročnost: Nízká

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P42 – Přidání frekvence placení k ceně v sumarizaci objednávky

V souhrnu objednávky bude u jednotlivých položek uvedeno, zdali bude cena účtována jednorázově či měsíčně.

Náročnost: Nízká

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P43 – Možnost filtrování objednávek

Uživateli bude umožněno filtrovat v seznamu zpracovaných objednávek, podobně jako je možné filtrovat logy.

Náročnost: Střední

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P44 – Úprava tabulky logů

Při filtrování podle spojení bude uživateli nabídnut seznam spojení, ze kterých si vybere to, dle kterého budou logy vyfiltrovány. Dále bude změněna ikona stavu záznamu synchronizace tak, aby se v uživateli neevokovala možnost kliknutí na danou ikonu. Také bude změněno rozbalení logu, které bude možné vyvolat kliknutím na celý řádek.

Náročnost: Nízká

Priorita: Could have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

P45 – Přidání tutorialu

Uživatel dostane po prvním přihlášení možnost zobrazení tutorialu, který ho provede aplikací a popíše mu základní funkce, konfiguraci spojení a nastavení účtu.

Náročnost: Vysoká

Priorita: Won't have

Dotčené části: Frontend

Typ požadavku: Funkční – funkčnost

Propagační web

Podobně jako v předchozí sekci požadavků, i následující vychází z bakalářské práce Bc. Jakuba Čermáka [15], pro které byly taktéž vytvořeny úkoly v projektovém nástroji Redmine. Jelikož se jedná pouze o propagační web, požadavky s ním související nijak neovlivňují funkčnost celého nástroje Timer2Ticket a z tohoto důvodu je priorita požadavků v současnosti velmi nízká.

P46 – Dokončení všech sekcí propagačního webu
Bude dokončen propagační web, to zahrnuje dokončení sekcí open-source, o projektu a kontakt. Za poslední sekcí bude zobrazeno zápatí společnosti Jagu s.r.o.

Náročnost: Střední

Priorita: Could have

Dotčené části: Propagační web

Typ požadavku: Funkční – funkčnost

P47 – Dokončení hlavičky propagačního webu
Bude dokončena hlavička propagačního webu. To zahrnuje přidání loga Timer2Ticket na levou stranu, dále zprovoznění všech tlačítek a přidání tlačítka pro přesměrování na kontaktní formulář.

Náročnost: Nízká

Priorita: Could have

Dotčené části: Propagační web

Typ požadavku: Funkční – funkčnost

P48 – Responzivní propagační web
Bude zajištěno, že všechny části propagačního webu budou responzivní. To zahrnuje i doplnění responzivity do již vytvořených částí webu.

Náročnost: Střední

Priorita: Could have

Dotčené části: Propagační web

Typ požadavku: Funkční – funkčnost

P49 – Přidání překladů
Propagační web bude podporovat překlady. Bude dostupný minimálně v jazyce českém a anglickém. Jazyk bude volen na základě nastavení prohlížeče. Nebude-li jazyk prohlížeče podporován, použije se anglický překlad. Uživatel bude mít možnost vybrat překlad manuálně.

Náročnost: Střední

Priorita: Could have

Dotčené části: Propagační web

Typ požadavku: Funkční – funkčnost

P50 – Úvodní video
Bude vytvořené video, které bude umístěno v úvodní části webu. Cílem videa je upoutání pozornosti. Video bude shrnovat základní funkce a principy aplikace.

Náročnost: Střední

Priorita: Could have

Dotčené části: Propagační web

Typ požadavku: Funkční – funkčnost

P51 – Tutorial videa

Budou vytvořeny videa, která seznámí uživatele s aplikací, přiblíží mu používání aplikace a vyjasní mu principy a celý koncept nástroje Timer2Ticket.

Náročnost: Střední

Priorita: Won't have

Dotčené části: Propagační web

Typ požadavku: Funkční – funkčnost

Kapitola 3
Návrh

S ohledem na požadavky, které vzešly z předchozí kapitoly, bude zapotřebí upravit aplikaci tak, aby s nimi byla ve shodě. Z tohoto důvodu dojde na několika místech k úpravám a změnám v logice nástroje Timer2Ticket. Poté bude rozšířena architektura aplikace a provedeny úpravy již existujících scénářů užití.

Vzhledem k provádění návrhu dle seznamu požadavků vytvořeném v kapitole Analýza, bude i text této kapitoly členěn dle jednotlivých logických celků obdobně tak, jako jsou členěny požadavky v tomto seznamu. Pouze v závěru této kapitoly budou popsány oblasti společné pro celý nástroj Timer2Ticket – databázový model, architektura a případy užití.

3.1 Požadavky ke zpracování

Jelikož je rozsah požadavků, které vzešly z analýzy značný, bylo po konzultaci s vedoucím práce z důvodu vysoké náročnosti některých požadavků rozhodnuto nerealizovat všechny požadavky. Tato práce se tedy bude zabývat pouze požadavky se stanovenou nejvyšší prioritou, tedy s prioritou Must have a téměř všemi s prioritou Should have. Jedinými požadavky s prioritou Should have, kterými se práce nebude zabývat z důvodu vysoké náročnosti, jsou požadavky P23–P25, tedy požadavky věnující se přidání nástroje Jira.

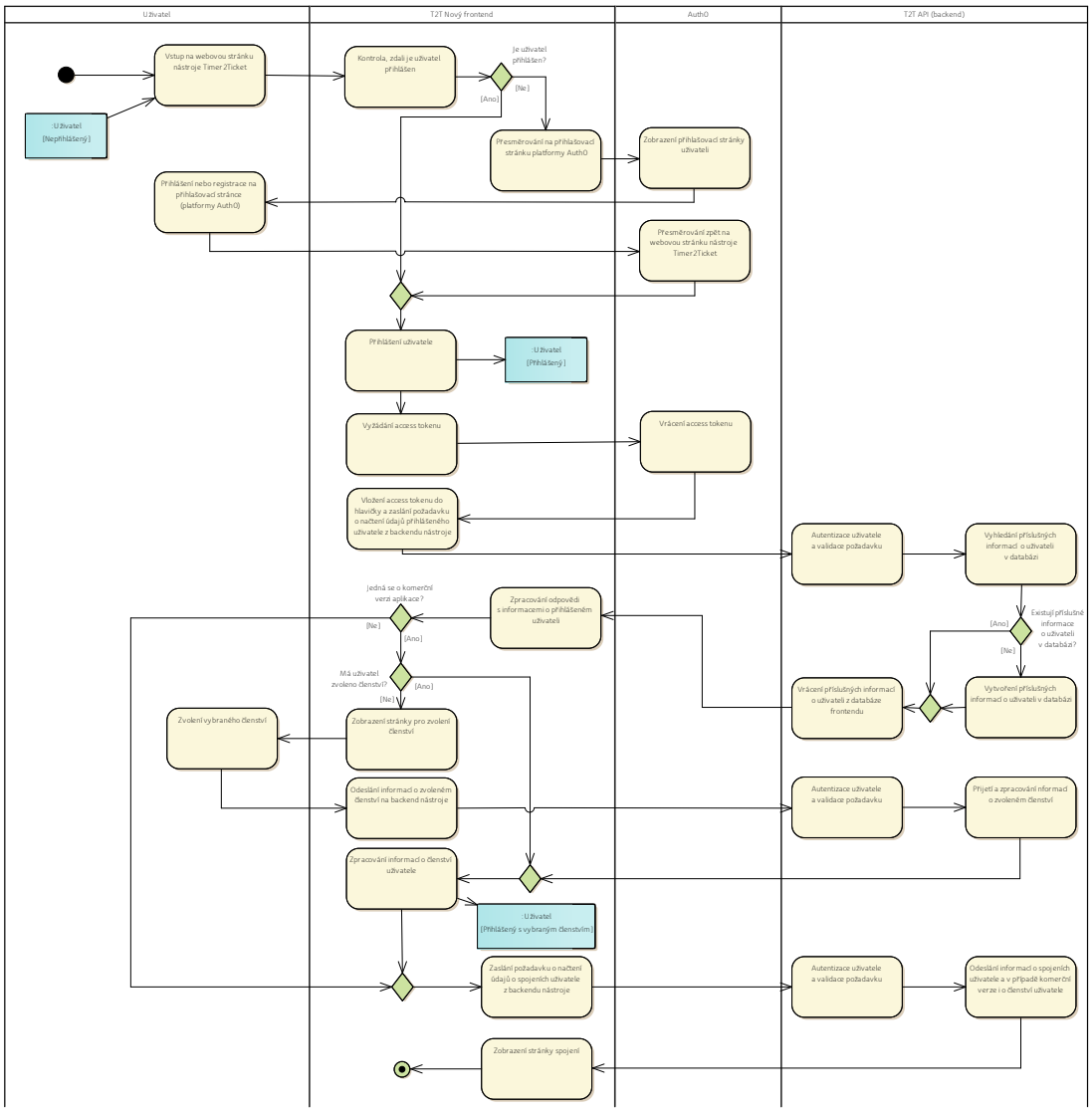
3.2 Napojení nového frontendu na backend

V souvislosti s touto skupinou požadavků se jedná především o rozšíření backendu tak, aby podporoval funkce implementované v novém frontendu v rámci bakalářské práce Bc. Jakuba Čermáka. Chování většiny požadavků tak již byly v této práci definovány. Pro tyto požadavky je tedy v souvislosti s návrhem relevantní především změna databázového modelu popsaná dále v kapitole. V této podkapitole je tak řešena registrace a přihlašování uživatelů a zobrazení logů.

Registrace a přihlášení uživatelů

Tato část textu bude zaměřena na návrh přihlašování, respektive registrace uživatelů, jejich autentizaci a synchronizaci dat mezi platformou Auth0, frontendem a backendem nástroje Timer2Ticket. Pro snazší pochopení následujícího textu je přiložen diagram aktivit, který lze nalézt na obrázku 3.1.

Při vstupu na novou webovou stránku nástroje Timer2Ticket je zkontrolováno, zdali je uživatel přihlášen. To bude vykonáno pomocí knihovny `auth0-spa-js` [71]. Nebude-li přihlášen, dojde k přesměrování na přihlašovací stránku platformy Auth0. Na této stránce se může uživatel



Obrázek 3.1 Diagram aktivit přihlášení uživatele do nástroje Timer2Ticket, vytvořeno pomocí [70], notace UML

přihlásit nebo zaregistrovat, nemá-li ještě uživatelský účet vytvořen. Po přihlášení (diagram předpokládá správně zadané přihlašovací údaje) dojde k přesměrování zpět na úvodní stránku Timer2Ticket. Toto chování je implementováno i v současné verzi a tak zůstane zachováno.

Po přesměrování z přihlašovací stránky zpět si frontend na pozadí vyžádá ze stejné knihovny JWT přístupový token, který bude následně vkládán do hlavičky `Authorization` každého požadavku, který vyžaduje autentizaci uživatele. Následně bude, stále na pozadí, zaslán HTTP požadavek na načtení údajů přihlášeného uživatele z databáze, respektive backendu. Po obdržení požadavku nejprve backend provede autentizaci uživatele a validaci požadavku. Pro autentizaci tokenu bude využita knihovna `express-oauth2-jwt-bearer`, který zkontroluje platnost obdrženého tokenu. Poté budou příslušná data vyhledána v databázi. Nebudou-li existovat, budou informace o uživateli vytvořena. Příslušné informace budou získány ze zaslaného JWT tokenu.

Obdržená data budou na frontendu zpracována. Jedná-li se o nekomerční verzi aplikace, budou z backendu získána data o spojeních uživatele a následně se uživateli zobrazí stránka spojení, čímž proces končí. V případě komerční verze aplikace však ještě bude zkontrolováno, zda má uživatel již zvolené členství. Pokud je členství zvoleno, chování je stejné jako v případě nekomerční verze aplikace. V případě zatím nezvoleného členství je uživateli zobrazena stránka pro výběr členství. Bude-li členství vybráno, data budou odeslána ke zpracování na backend. Po vybrání členství již uživateli nic nebrání začít využívat nástroj Timer2Ticket.

V této podkapitole nebylo pro přehlednost zahrnuto uhrazení poplatku za zvolené členství. Tento případ je popsán dále v textu.

3.2.1 Zobrazení logů

Pro požadavek P5 je zapotřebí upravit především vytváření objektu logu jobu spojení. Bude-li se jednat o automatický job, logika měněna nebude. V případě okamžité synchronizace vyvolané uživatelem však bude nutné vytvářet tento objekt již v části API po zavolání příslušného koncového bodu pro okamžitou synchronizaci. Hlavním důvodem je v případě komerční verze vytváření záznamu použité okamžité synchronizace, který obsahuje odkaz na objekt logu jobu spojení. Tento záznam totiž musí být k zobrazení ihned po provedení akce a nezle čekat, až bude objekt vytvořen částí T2T Core.

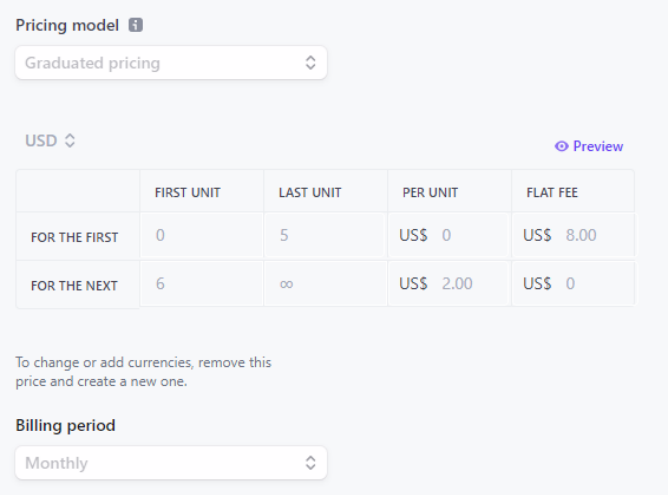
ID objektu logu jobu spojení bude následně zasláno z části T2T API do části T2T Core pro vykonání okamžité synchronizace. T2T Core následně nalezne objekt s daným ID v databázi, ze kterého získá ID spojení, které má být synchronizováno. Poté již nic nebrání vykonání synchronizačního jobu.

3.3 Zprovoznění monetizace

Tato podkapitola se zabývá návrhem požadavků P11 – P20, které se týkají zprovoznění monetizace nástroje Timer2Ticket. Především zde bude popsána konfigurace platební brány Stripe tak, aby vyhovovala monetizačnímu modelu nástroje Timer2Ticket a úpravy ve zdrojových kódech backendu i frontendu.

3.3.1 Úvodní konfigurace platební brány Stripe

Před zahájením veškerých prací bude potřeba vytvořit si na webové stránce platební brány Stripe uživatelský účet. Po přihlášení bude třeba vytvořit tzv. account, tedy oddělený prostor pro jednu nezávislou oblast podnikání. Jeho vytvořením se následně zobrazí tzv. dashboard, tedy webové prostředí správy a konfigurace platební brány. Platební bránu lze však v tomto okamžiku využívat pouze v testovacím režimu. Pro aktivaci skutečných plateb je nutné zadat několik informací týkající se podnikání, jako je typ a místo podnikání, informace pro ověření identity nebo bankovní spojení. [50]



Obrázek 3.2 Ukázka nastavení cenového modelu platební brány Stripe pro členství senior

Vytvoření produktů

Nutné je taktéž vytvořit produkty, kdy v případě Timer2Ticket jde o nabízené členství a přikoupení aktivních spojení nad rámec členství pro opakované platby a jednorázovou koupi okamžitých synchronizací.

Pro vytvoření produktů členství stačí vložit název členství, popis a zvolit cenový model „graduated pricing“. Tento cenový model nejlépe odpovídá potřebám Timer2Ticket – umožňuje výběr členství a možnost zadat v případě nástroje Timer2Ticket počet aktivních spojení, tudíž i spojení nad rámec členství. Rozhodnutí, že se počet maximálních aktivních spojení, a tedy i spojení nad rámec členství budou volit zadáním množství při koupi členství, bylo vykonáno po konzultaci s Ing. Oldřichem Malcem. Vzhledem k původní implementaci nového frontendu Bc. Jakubem Čermákem [15], který uváděl ceny v amerických dolarech, bylo zobrazování cen v této měně zachováno.

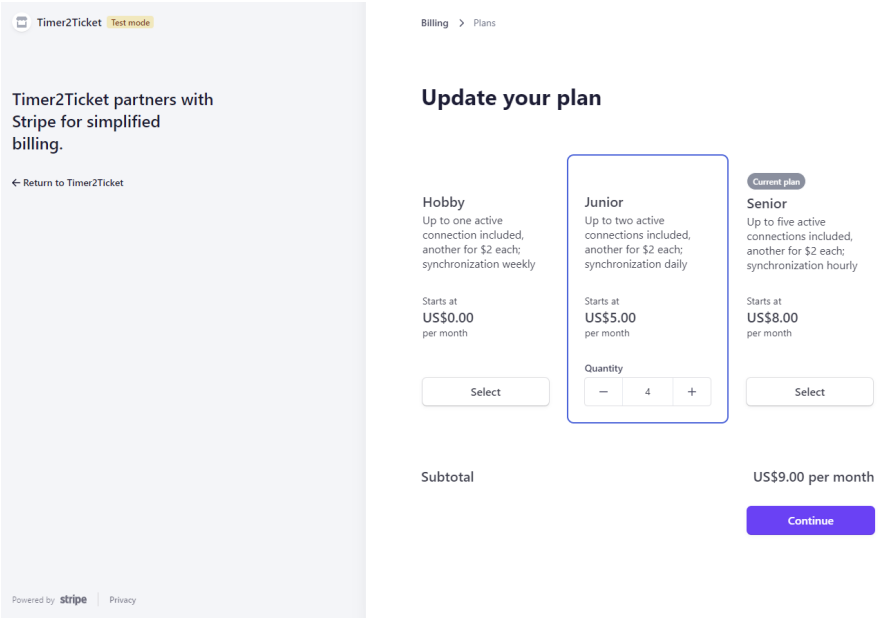
V tomto cenovém modelu budou nastaveny dvě cenové úrovně. První úroveň bude zahrnovat maximální možný počet aktivních spojení a „flat fee“, tedy základní cena, bude odpovídat ceně daného členství. Druhá úroveň pak bude pro přikoupení aktivních spojení nad rámec členství. Zde je nutné stanovit cenu „per unit“, tedy za jedno zakoupené aktivní spojení nad rámec členství. Pro ukázku je na obrázku 3.2 zobrazeno nastavení cenového modelu pro členství senior, který stojí 8 $. Za tuto cenu je uživateli umožněno mít až 5 aktivních spojení. Chce-li si připlatit za další spojení nad rámec členství, za každý zaplatí navíc 2 $.

Pro jednorázovou koupi okamžitých synchronizací je nastavení ceny poněkud jednodušší. Cenovému modelu tohoto produktu nejlépe odpovídá „package pricing“, tedy stanovení ceny za počet jednotek. Zde se zvolí pouze cena za jeden balíček a počet jednotek v balíčku. Jelikož se nejedná o opakovanou platbu, jako typ ceny je zvolen „one time“.

Konfigurace zákaznického portálu

Poté je třeba nakonfigurovat zákaznický portál, tedy webovou stránku, na které budou uživatelé moci spravovat své informace související s placením za službu Timer2Ticket. Na správu zákaznického portálu se lze dostat z dashboardu kliknutím na ozubené kolečko v pravém horním rohu obrazovky, kde se v sekci „Billing“ nachází možnost „Customer portal“.

Na následující stránce lze nastavit, co se zobrazí uživateli. Timer2Ticket bude umožňovat zobrazení faktur (Invoices → Invoice history), uživatelských informací, kde je třeba zaškrtnout všechny políčka kromě „Shipping address“. Následně je nutné nakonfigurovat předplatné. V této



Obrázek 3.3 Ukázka změny členství uživatele v zákaznickém portálu platební brány Stripe

sekci je potřeba zvolit, zdali uživatelé mohou měnit svůj plán, včetně množství (právě kvůli přikoupení aktivních spojení nad rámec členství). Následně jsou vybrány produkty, které budou uživateli při změně plánu zobrazeny. U každého je však nutné změnit pravidla pro množství, kdy se minimální množství rovná počtu aktivních spojení v ceně členství.

Zadání minimálního množství znamená, že si uživatel nemůže zakoupit menší množství aktivních spojení, než které poskytuje dané členství. Může si však přikoupit další, kdy za každý v současném nastavení zaplatí navíc 2 $. Tato konfigurace tak odpovídá monetizačnímu modelu nástroje Timer2Ticket.

Na obrázku 3.3 je znázorněna změna členství uživatele v zákaznickém portálu platební brány Stripe.

Nastavení daně

Do nastavení daně se lze dostat z dashboardu kliknutím na „More“ v horním menu a následným výběrem možnosti „Tax“. Není-li automatický výpočet daní zatím nastaven, lze tak učinit kliknutím na „Get started“ a poté na „Tax settings“. Na této stránce je nutné zadat zemi, ve které odvádí provozovatel daně. Následuje nastavení výchozí kategorie daně produktů a zdali je daň zahrnuta v ceně nebo nikoliv. Jako poslední lze určit, zdali bude použito automatického výpočtu daně.

Dále je ještě nutné v záložce „Registrations“ zvolit zemi registrace u daňového úřadu, pokud podnikatel překročí daňovou hranici stanovenou zákonem. [72]

3.3.2 Vytvoření a úprava členství

V této sekci textu bude vytvořen návrh pro požadavek P11 týkající se členství uživatele a požadavek P15 popisující jeho okamžité ukončení. S členstvím uživatele souvisí možnost koupě aktivních spojení nad rámec členství, popsaná v požadavku P12.

Timer2Ticket bude přebírat informace o členství uživatele z platební brány Stripe. Nebude-li mít uživatel členství, znamená to, že si ho od registrace zatím nezvolil, nebo se nezdařila opako-

vaná platba a členství tak bylo zrušeno. V tomto případě se mu na webové stránce Timer2Ticket zobrazí tabulka s výběrem členství. Ta se již nachází v současné implementaci nového frontendu. Dojde však k jejímu rozšíření tak, že i v tomto kroku bude mít uživatel možnost přikoupení aktivních spojení nad rámec členství.

Po výběru členství bude následně přesměrován na stránku platební brány Stripe. kde bude vyzván k zadání fakturačních údajů. Fakturační údaje je nutné uvádět z důvodu daňových povinností provozovatele. V případě nutnosti úhrady bude uživatel vyzván i k zadání platebních údajů. Po vyplnění formuláře dojde k přesměrování zpět na webovou stránku Timer2Ticket.

Jelikož budou zadávány fakturační údaje přímo do platební brány Stripe, nebude nutná vlastní implementace ekvivalentního formuláře, která se nachází v současné implementaci. Odstraněním vlastní implementace tohoto formuláře dojde k vyřešení požadavku „P39 – Úprava formuláře s platebními údaji“, jelikož již nebude nutné tento formulář upravovat. Jelikož Stripe umožňuje zapamatování zadaných údajů, je tímto krokem i částečně vyřešen požadavek P20.

K úpravám uživatelského rozhraní dojde i při implementaci změny členství uživatele. V podsekci členství sekce profil si již uživatel nebude volit nové členství ani počet spojení nad rámec spojení, ale taktéž dojde k přesměrování na formulář v platební bráně Stripe. Zde bude možné současné členství změnit nebo upravit, včetně změny počtu spojení nad rámec členství. Bude-li si přát uživatel okamžitě ukončit současné placené předplatné, dosáhne toho změnou členství na základní, hobby členství s jedním aktivním spojením, které je zdarma. Při změně členství bude vystavena faktura okamžitě při změně předplatného, v případě povýšení tak bude okamžitě uhrazen rozdíl v ceně, případně aplikována sleva na budoucí faktury.

Logika změny členství

V souvislosti se změnou členství, respektive při přechodu na nižší úroveň členství, bude nutné upravit konfiguraci spojení uživatele. Snížením členství se totiž může změnit maximální počet aktivních spojení nebo četnost synchronizace objektů. Změna maximálního počtu aktivních spojení pak může nastat i změnou počtu přikoupených aktivních spojení nad rámec členství. Je tedy nutné navrhnout, jak se v těchto případech zachovat. Při povýšení členství tento problém nenastává, změnu četnosti spojení nebo aktivaci nových může uživatel provést v uživatelském rozhraní aplikace.

Tato oblast byla diskutována se zadavatelem na jedné z konzultací. Z návrhů vzešlo jako nejoptimálnější nechat uživateli jistý čas, například 3 dny po snížení členství, aby si upravil svá spojení dle parametrů nově zvoleného členství. Po uplynutí tohoto období by pak byla změna provedena automaticky, pokud by uživatel svá spojení korektně neupravil.

Implementace tohoto chování by však byla poměrně náročná, z tohoto důvodu se dospělo ke kompromisu. Spojení budou automaticky upravena na základě parametrů nově zvoleného členství ihned po obdržení notifikace o změně členství z platební brány Stripe. Uživatel bude o změnách spojení informován e-mailem.

Sníží-li se maximální počet aktivních spojení a aktivní spojení uživatele budou tento limit přesahovat, spojení budou deaktivována od nejstaršího, dokud nebude dosaženo nového limitu. Při změně četnosti limitu může dojít ke třem případům. V případě přechodu z členství senior na junior bude pouze deaktivována synchronizace každou hodinu. Synchronizace pak bude probíhat v zadaný čas. Dny, ve kterých bude probíhat synchronizace, se nezmění.

Bude-li členství změněno z junior na hobby, budou změněny dny, ve kterých bude probíhat synchronizace tak, že se vezme z původního výčtu dnů nejdřívější den v týdnu. Změní-li se členství ze senior na hobby, provedou se oba druhy změn popsané výše. Informování uživatele e-mailem bude zachováno.

3.3.3 Uložení platebních údajů

Jak bylo zmíněno v předchozí sekci, zodpovědnost za ukládání platebních údajů bude převedena na platební bránu Stripe. Díky tomu nebude nutný v podsekci členství sekce profil box pro zadání fakturačních údajů, který tedy bude odstraněn. Stejně tak bude stejný formulář odstraněn i z podsekce platby. V této podsekci dojde k nahrazení formuláře tlačítkem, které přesměruje uživatele na zákaznický portál platební brány Stripe. Zde si uživatel bude moci změnit platební a fakturační údaje. Tímto dojde k naplnění požadavku P20 – Uložení platebních údajů.

3.3.4 Historie objednávek a stažení faktur

V zákaznickém portálu si uživatel bude moci zobrazit i historii objednávek, včetně stažení faktur. Tímto tedy budou vyřešeny požadavky P14 a P18. Přesunutí těchto funkcionalit do zákaznického portálu platební brány Stripe bylo konzultováno s Ing. Oldřichem Malcem a Bc. Jakubem Čermákem a je realizováno z důvodu snížení nutnosti správy a údržby zdrojového kódu těchto funkcionalit a prevence možných chyb.

V souvislosti s těmito požadavky bude nutné upravit část podsekce platby sekce profil věnující se zpracovaným objednávkám, kde již nebude zobrazena tabulka s možností stažení faktury, ale pouze tlačítko, které podobně jako při změně fakturačních údajů přesměruje uživatele na zákaznický portál platební brány Stripe.

Přesunutím seznamu zpracovaných objednávek do zákaznického portálu se tak nebude nutné zabývat požadavkem „P43 – Možnost filtrování objednávek“.

3.3.5 Podpora okamžitých synchronizací

Podpory okamžitých synchronizací se týká požadavek P13. Ta samotná je již obsažena jak v původní implementaci Ing. Víta Štefana, tak i v novém frontendu Bc. Jakuba Čermáka. Zbývá tedy vyřešit způsob, jakým si uživatelé budou moci okamžité synchronizace zakoupit.

Zakoupení okamžitých synchronizací bude poměrně přímočaré. Uživatel si na webové stránce Timer2Ticket v boxu okamžité synchronizace v podsekci členství sekce profil zvolí počet okamžitých synchronizací, který si přeje zakoupit. Nákup však již nebude spojen se změnou předplatného, ale bude prováděn samostatně. Tím dojde ke striktnímu oddělení opakované a jednorázové platby. Tímto rozhodnutím dojde k vyřešení požadavku P42, kdy by uvádění četnosti účtování jednotlivých položek postrádala informační hodnotu.

Parametry budou po stisknutí tlačítka potvrzující zadaný počet okamžitých synchronizací odeslány na backend, který pomocí Stripe API [49] vygeneruje URL pro uhrazení, na kterou bude následně uživatel přesměrován. Po úspěšné platbě bude přesměrován zpět na uživatelské rozhraní Timer2Ticket.

3.3.6 Logování použitých okamžitých synchronizací

Záznam o použití okamžité synchronizace bude vytvářen v komerční verzi aplikace po vyvolání okamžité synchronizace daného spojení. V záznamu bude uložena informace o spojení, pro které byla okamžitá synchronizace využita, respektive informace o zakoupení okamžitých synchronizací.

Informace o spojení bude ve formě textového řetězce, aby v případě změny konfigurace spojení byla zachována informace o původní konfiguraci spojení. Záznam taktéž nebude odkazovat přímo na objekt spojení, to právě z důvodu ukládání informací o spojení ve formě textu. V případě potřeby však bude obsahovat odkaz na log jobu, přes který bude možné detailní informace o spojení získat. V této práci však tato možnost nebude využita.

3.3.7 Napojení platební brány

Požadavek „P17 – Napojení platební brány“ řeší propagaci informací, respektive změn provedených uživatelem, do nástroje Timer2Ticket. Pro získání informací o provedené akci v platební bráně Stripe (tedy změna členství nebo zakoupení okamžitých synchronizací) bude využito webhooků. Ty používá Stripe k upozornění aplikace, že proběhla nějaká událost. Touto cestou budou předána data backendu, respektive části T2T API, která příchozí zprávu zpracuje. [50]

3.3.8 Převod faktur do nástroje Fakturoid

Po úspěšném uhrazení faktury v platební bráně Stripe bude vytvořena totožná faktura v nástroji Fakturoid, čímž dojde k naplnění požadavku P19. To bude realizováno tak, že je po úhradě zaslán webhook. Událost následně bude odchycena v části T2T API, kde dojde k jejímu zpracování.

Tato událost obsahuje všechny potřebné náležitosti nutné k vytvoření faktury ve Fakturoidu. Těmi jsou informace o zákazníkovi a informace o zakoupených položkách, včetně uhrazené ceny a částky daně. Po převedení dat z obdržené události do formátu, který přijímá Fakturoid, bude vytvořen a zaslán příslušný požadavek dle API dokumentace nástroje Fakturoid pro vytvoření faktury. Jelikož nástroj neposkytuje knihovnu, bude nutné vytvořit HTTP požadavek manuálně a ten následně zaslat s využitím knihovny superagent.

3.4 Oddělení komerční a nekomerční části

V tomto logickém celku se nachází pouze jeden požadavek – P22 – Oddělení částí zdrojových kódů. Ve zdrojových kódech bude nutné oddělit co nejvíce částí, které slouží pro účely komerční verze. Z tohoto důvodu budou vytvořeny dvě knihovny, které budou používány v případě komerční verze aplikace. Knihovny budou rozšiřovat open-source verzi, ta však bude funkční, i pokud nebudou správcem balíčků npm nainstalovány.

Jedna knihovna bude určena pro nový frontend, která bude obsahovat komponenty použité při běhu komerční verze. Dále bude poskytovat funkce, které je nutné volat i z nekomerční části, tedy z komponent, které nejde vyčlenit do této knihovny.

Druhá knihovna pak bude využívána backendem, respektive částí T2T API. Předpokládá se, že tato knihovna bude především poskytovat funkce a metody pro validaci konfigurace spojení tak, aby odpovídala zvolenému členství uživatele. Dále bude zajišťovat interakci s platební bránou Stripe a nástrojem Fakturoid.

Část T2T Core bude totožná pro jak komerční, tak i nekomerční část, jelikož se tato část stará pouze o provádění synchronizací dle plánu synchronizace, respektive při použití okamžitých synchronizací. Za správný plán synchronizace zodpovídá část T2T API, která i posílá požadavek na provedení okamžité synchronizace. Proto se v části T2T Core nenachází zdrojový kód využívaný pouze v komerční části aplikace a tak ani není nutné vytvářet této části knihovnu oddělující části zdrojového kódu.

3.5 Zpětná kompatibilita

Jak vyplývá z analytické části, konkrétně z požadavku P26 – Migrace původní verze Timer2Ticket na novou verzi, zpětná kompatibilita bude převedením informací o uživatelích a jejich nakonfigurovaných spojeních do nové verze nástroje Timer2Ticket.

Zpětná kompatibilita s původní verzí nástroje Timer2Ticket bude realizována vytvořením migračního skriptu, který převede data uživatele z původního objektového modelu do nově vytvořeného. Při implementaci tohoto skriptu bude využito nové databázové schéma, které je ob-

saženo v podkapitole Databázový model na obrázku 3.4. Tato podkapitola taktéž obsahuje popis provedených změn v databázovém modelu, dle kterého bude migrační skript realizován.

Migrace bude realizována jako jednosměrná. To je dostatečné, jelikož se v budoucnu plánuje přechod na novou verzi nástroje Timer2Ticket a tak není a nebude důvod data migrovat zpět z nové verze na původní. Migrace bude probíhat tak, že budou najednou zmigrováni všichni uživatelé. To bude třeba udělat při přechodu z jedné verze systému na druhou. Přenasazení a provedení migrace je vhodné provést najednou, jelikož při používání obou verzí najednou po provedení migrace dat může vést k duplicitám, kdy obě verze systému budou pro nově vzniklé časové záznamy vytvářet vlastní mapovací objekty. Důsledkem toho pak bude, že synchronizované časové záznamy budou synchronizovány dvakrát. Tento postup vzešel po konzultaci se zadavatelem práce a jeho následným odsouhlasením.

Jelikož bude migrační skript použit pravděpodobně pouze jednou, neboť migrace dat bude jednorázová věc, bude rozhodnuto pro jeho realizaci vytvořit samostatný projekt. Migrační skript tak nebude součástí zdrojových kódů nástroje. Migrační skript bude, stejně jako celý backend nástroje Timer2Ticket, napsán v jazyce TypeScript. Toto rozhodnutí umožňuje snadné využití již implementovaných tříd z ostatních projektů obsahující části backendu.

3.6 Databázový model

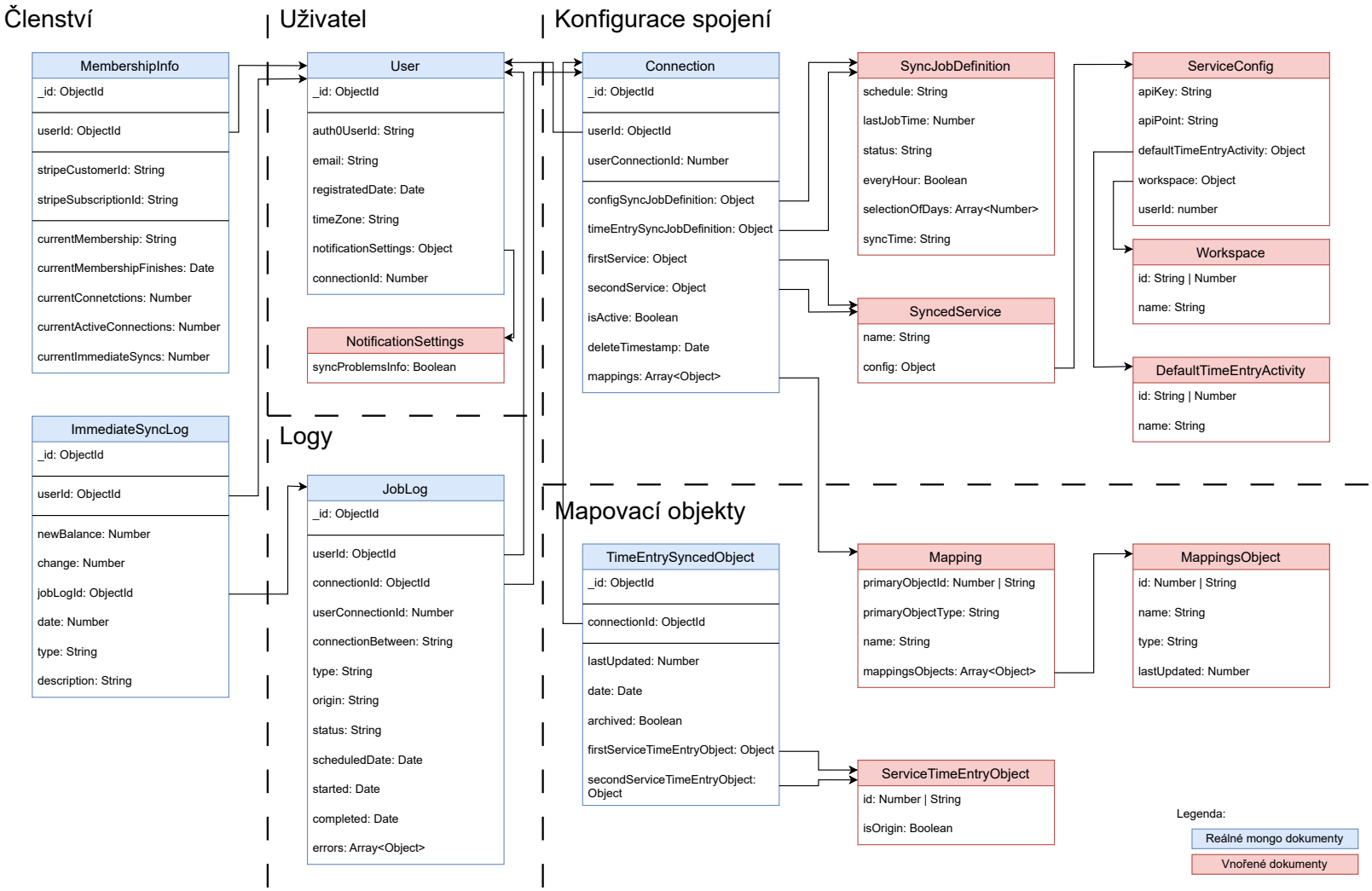
Vzhledem k poměrně velkým změnám, které přinesla do nástroje Timer2Ticket implementace prototypu nového frontendu, bude nutné výrazně upravit backendovou část, včetně databázového modelu. Pro snazší orientaci v dále popisovaných změnách je přiloženo schéma databázového modelu, které je možné nalézt na obrázku 3.4. Na něm jsou pro snazší orientaci dokumenty členěny do větších logických celků.

Pro znázorněný databázový model nebyla použita žádná standardizovaná notace. Jelikož je použita NoSQL databáze MongoDB, nebylo možné použít relační schéma. Ustoupeno bylo i od použití konceptuálního schématu, ve kterém by bylo obtížné zachytit vnořené dokumenty či odkazy na jiné dokumenty pomocí `ObjectId`.

Jelikož je MongoDB dokumentová databáze ukládající data ve formátu BSON [73], umožňuje vkládání serializovaných instancí tříd používaných při běhu aplikace do kolekcí. Z tohoto důvodu se databázové schéma shoduje s třídami, které jsou používány ve zdrojovém kódu backendu.

Schéma předpokládá existenci tří kolekcí pro nekomerční, respektive pěti pro komerční verzi nástroje – pro uživatele, spojení, logy provedených jobů a dále informace o členství uživatelů spolu s historií provedených okamžitých synchronizací v komerční verzi.

Změny jsou popsány v následujících podsekcích, členěny stejně jako dokumenty v databázovém schématu.



Obrázek 3.4 Návrh nového databázového schématu pro nástroj Timer2Ticket, vytvořeno pomocí [74], bez notace

Uživatel

Prvním popisovaná kolekce pro ukládání dokumentů `User` reprezentující uživatele. V tomto dokumentu bylo provedeno několik významných změn. První je odstranění atributů přihlašovacích údajů – `username` a `passwordHash`. Ty již nejsou třeba, jelikož přihlašování uživatelů bylo kompletně nahrazeno platformou Auth0. Místo toho však přibyl atribut `auth0UserId` odkazující právě na tuto platformu. Tento atribut udržuje odkaz do Auth0 pro spárování instance uživatele v databázích Timer2Ticket a Auth0.

Místo atributu `username`, který se rovnal zadanému e-mailu, je přidán atribut `email`. Nejedná se však o pouhé přejmenování, jelikož e-mail nemusí být zadán, pokud se uživatel přihlásí přes sociální síť, která e-mail nevyžaduje. Tento atribut slouží k zasílání e-mailových notifikací.

Atribut `registrated` byl přejmenován na `registratedDate`. `status` byl odebrán úplně, jelikož v nové implementaci frontendu již není potřeba.

Odebrán, respektive přesunut do dokumentu `MembershipInfo` byl vnořený dokument obsažen v atributu `config`. Ten obsahuje atribut `plan`, který nově bude reprezentován zvoleným členstvím uživatele a `daysToSync` nebude v nové implementaci potřeba.

S konfigurací spojení souvisí čtyři původní atributy, které byly taktéž přesunuty, a to do dokumentu `Connection`: `serviceDefinitions`, `configSyncJobDefinition`, `timeEntrySyncJobDefinition` a `mappings`.

Posledními nepopsanými atributy jsou `notificationSettings`, `connectionId` a `timeZone`. Prvním z nich je vnořený dokument, ve kterém jsou nově obsaženy informace o typu zasílaných notifikací na e-mailovou adresu uživatele. `connectionId` slouží pro přiřazování ID spojením v rámci jednoho uživatele, kdy je v tomto atributu udržována hodnota následujícího ID spojení v rámci jednoho uživatele. Toto ID je pak uživateli zobrazováno v přehledu spojení v novém frontendu.

Jelikož se v kolekci uživatelů často vyhledává kromě implicitního atributu `_id`, kterému je index vytvořen automaticky, i pomocí atributu `auth0UserId`, je vhodné vytvořit vyhledávací index i nad tímto atributem.

Konfigurace spojení

Tento logický celek obsahuje taktéž pouze jednu MongoDB kolekci, a to pro dokument `Connection` reprezentující jedno vytvořené spojení uživateli. Tento dokument není obsažen v předchozí verzi, jelikož ta neumožňovala vytvářet více spojení. Informace o spojení byly uloženy přímo v dokumentu `User`.

Identifikátor uživatele, jemuž spojení náleží, se nachází v atributu `userId`. ID spojení v rámci uživatele, který umožňuje snazší práci uživatele se spojeními v uživatelském rozhraní aplikace, reprezentuje atribut `userConnectionId`.

Následují atributy `configSyncJobDefinition`, `timeEntrySyncJobDefinition`, jež jsou totožné jako v původní verzi, jen byly přesunuty z dokumentu `User`. Tyto atributy obsahují vnořený dokument `SyncJobDefinition`, jemuž zůstaly původní atributy pro plán synchronizace definovaný cron výrazem a čas posledního synchronizačního jobu. Pro účely snadného zobrazování dat v novém frontendu však ještě byly přidány atributy `status` pro výsledek posledního synchronizačního jobu, jenž může nabývat následujících hodnot dle stavu jobu: `SUCCESS`, `ERROR`, `IN_PROGRESS` nebo `null`, nebyl-li zatím žádný synchronizační job spuštěn.

Dalšími atributy jsou `everyHour` indikující, zdali má synchronizace probíhat každou hodinu, `syncTime` pro čas synchronizace a `selectionOfDays` obsahující pole hodnot pro zvolené dny, kdy má probíhat synchronizace. Z těchto hodnot bude při vytváření objektu v implementaci odvozen cron výraz pro plán synchronizace. Obsah atributu `mappings` zůstal beze změn.

Posledním přesunutým atributem je `serviceDefinitions`, pole obsahující konfiguraci synchronizovaných služeb. Jelikož bude spojení synchronizovat data pouze mezi dvojicí nástrojů, došlo ke zjednodušení tak, že byl tento atribut nahrazen dvojicí vnořených kolekcí obsahující stejná data: `firstService` a `secondService`. Následující vnořené objekty v této sekci nebyly výrazně změněny,

pouze původní atribut `ServiceDefinition` byl přejmenován na `SyncedService` a atribut `apiKey` byl přesunut do dokumentu `ServiceConfig`.

Dokument dále obsahuje atribut `isActive` říkající, zdali je spojení aktivní a tedy dochází k synchronizaci údajů, či nikoliv. Posledním atributem je pak `deleteTimestamp`, který bude použit v případě mazání spojení. Jelikož nový frontend umožňuje vrátit smazané spojení, není vhodné z databáze spojení okamžitě smazat. Z tohoto důvodu byl přidán tento atribut, ve kterém je uložena doba smazání spojení. Pokud spojení nebude obnoveno, bude po dvou dnech smazáno, podobně jako jsou mazány logy jobů. Spolu se spojením budou smazány i dokumenty `TimeEntrySyncedObject` náležící smazanému spojení.

Jelikož jsou spojení vyhledávána nejen dle implicitního `_id`, ale i podle uživatele, kterému náleží, bude nutné vytvořit index nad atributem `userId`.

Mapovací objekty

Jak již bylo zmíněno výše, dokumenty `Mapping` a `MappingsObject`, jež jsou obsaženy v atributu `mappings` dokumentu `Connection`, pochází z původní verze nástroje Timer2Ticket. Beze změny zůstaly i dokumenty `TimeEntrySyncedObject` (kromě změny odkazu, nyní je udržován odkaz na spojení oproti původní verzi odkazující na uživatele), respektive `ServiceTimerEntryObject` reprezentující mapovací objekty synchronizovaných časových záznamů.

Logy

V tomto logickém celku je obsazena pouze jedna kolekce obsahující dokumenty `JobLog`, reprezentující logy jobů uživatele. Tato kolekce zůstala téměř beze změn, doplněny byly pouze tři atributy.

První doplněný atribut `connectionId` odkazuje na spojení, jemuž log náleží. Následuje atribut `userConnectionId`, který slouží pouze pro snazší dotazování a obsahuje ID spojení v rámci uživatele. Jinak by šel získat pomocí předchozího atributu. Posledním přidaným atributem je pak `connectionBetween`, které obsahuje textovou informaci o službách, mezi nimiž job provedl synchronizaci. Atribut byl přidán z důvodu, že se může měnit konfigurace služeb, mezi nimiž probíhá synchronizace, a tak případné získání této informace s využitím atributu `connectionId` by tak nemusela být správná. Uložení informace o službách do samostatného atributu též předchází případnému smazání spojení a tak nezobrazování dat z dokumentu `Connection`, který již nemusí existovat.

Logy jsou ve zdrojovém kódu vyhledávány dle implicitního `_id`. Při zobrazování tabulky logů na frontendu je nutné též vyhledat logy dle uživatele, jemuž logy náleží. Z tohoto důvodu je vhodné vytvořit vyhledávací index nad atributem `userId`.

Členství

Poslední popsaný logický celek členství obsahuje dvě kolekce dokumentů, které budou využívány pouze v případě komerční verze aplikace. První je kolekce pro dokumenty `MembershipInfo` reprezentující informaci o členství uživatele. Tento dokument bude vytvořen během zpracovávání registrace uživatele, nemůže se tedy stát, že nebude uživateli vytvořen.

I tento dokument obsahuje identifikátor uživatele, jemuž náleží informace o členství, a to v atributu `userId`. Podobně jako u předchozích kolekcí, i nad tímto atributem je vhodné vytvořit vyhledávací index.

Aby byly informace o uživatelském účtu správně synchronizovávány s daty z platební brány, je potřeba uchovávat dva atributy: `stripeCustomerId` a `stripeSubscriptionId`. První udržuje informace o datech uživatele v platební bráně Stripe a bude sloužit například pro správné zpracovávání

a přiřazení dat příchozích webhooků z platební brány nebo chce-li si uživatel skrze zákaznický portál zobrazit své zpracované objednávky či změnit členství.

Atribut `stripeCustomerId` bude při vytvoření uživatele nastaven na výchozí hodnotu `null`. Po výběru prvního členství bude před přesměrováním na stránku platební brány vytvořen uživatelský účet, jehož ID bude do tohoto parametru uloženo. Následně se již nebude měnit.

Druhý atribut `stripeSubscriptionId` bude obsahovat ID členství v platební bráně v případě, že má uživatel platné členství. V opačném případě bude `null` – to nastane buď po prvním přihlášení uživatele, kdy si ještě nezvolil členství, nebo v případě neúspěšné opakované platby za členství, kdy bude členství zrušeno.

Následují informace o členství uživatele, respektive atributů souvisejících s komerční verzí aplikace. Atribut `currentMembership` reprezentuje současné členství uživatele, datum, kdy členství vyprší, je uloženo v `currentMembershipFinishes`, `currentConnections` značí maximální počet aktivních spojení uživatele, `currentActiveConnections` počet v danou chvíli aktivních spojení a poslední atribut `currentImmediateSyncs` určuje zůstatek okamžitých synchronizací, tedy kolik jich ještě uživatel může využít, než si bude muset přikoupit další.

Druhá kolekce obsahuje dokumenty `ImmediateSyncLog`, které reprezentují záznam o použití okamžité synchronizace. Ta kromě implicitního atributu `_id` a odkazu na uživatele `userId`, jemuž záznam náleží, obsahuje informace o změně počtu okamžitých synchronizací.

Atributy `change` a `newBalance` obsahují informace o změně počtu okamžitých synchronizací a jejich nový zůstatek. `date` poskytuje informaci o čase, kdy ke změně došlo. Atribut `type` pak určuje typ záznamu. V budoucí implementaci budou tři typy těchto záznamů: `USE_TIME_ENTRIES`, `USE_CONFIG` a `BUY`. Z názvu vyplývá, že první dvě kolekce určují typ použití okamžité synchronizace. Třetí je určena pro zaznamenání přikoupení a tak nabytí počtu okamžitých synchronizací.

Atribut `jobLogId` bude obsahovat odkaz na log jobu, který byl touto okamžitou synchronizací spuštěn v kolekci `JobLog`. Jak je patrné z předchozí věty, tak tento atribut bude vyplněn pouze v případě, bude-li okamžitá synchronizace použita.

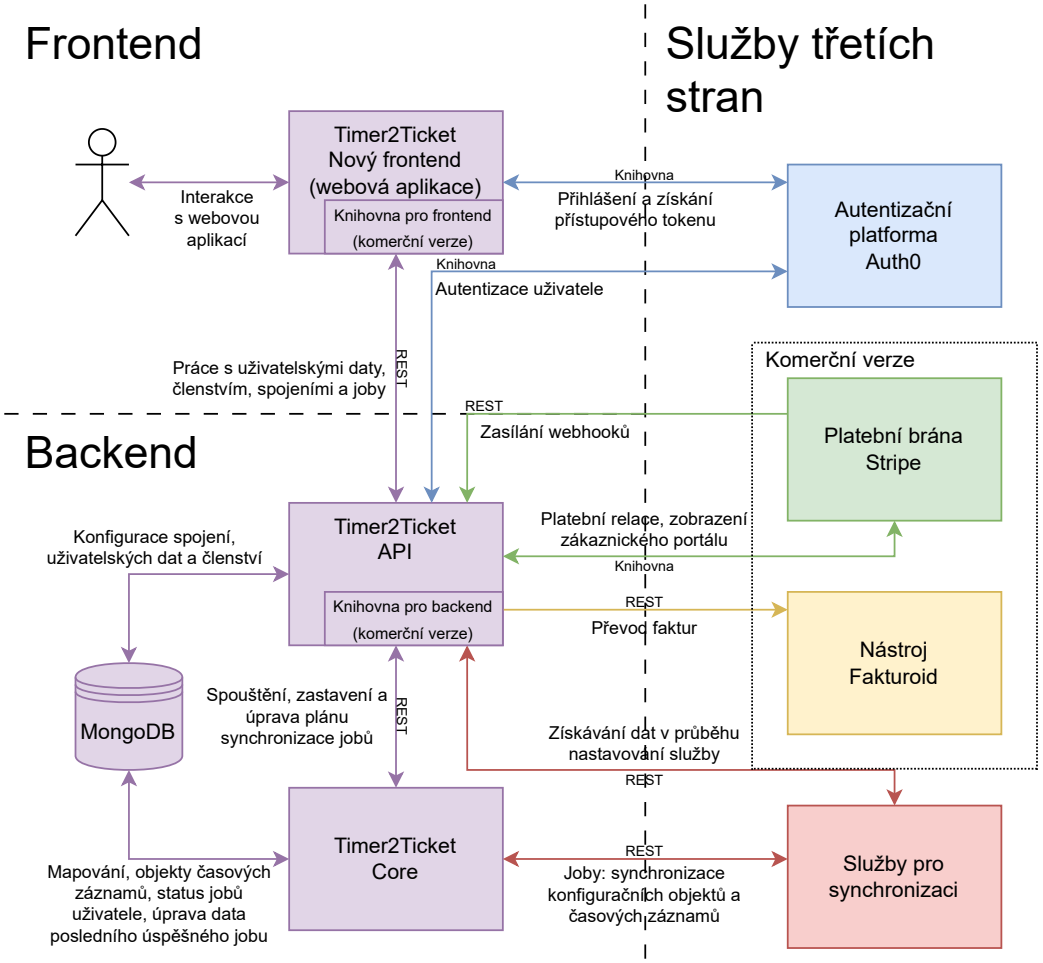
Posledním atributem je pak popis záznamu, tedy `description`. Ten bude v případě použití okamžité synchronizace obsahovat informace o spojení, které bylo synchronizováno. Důvod obsažení informace o spojení v samostatném atributu, nikoliv získání informací z atributu `jobLogId`, je ze stejného důvodu, jako je tomu u atributu `connectionBetween` dokumentu `JobLog`, tedy z důvodu možné změny konfigurace spojení a tak zobrazení potenciálně nesprávných informací o službách, mezi nimiž synchronizace proběhla. V případě přikoupení okamžitých synchronizací budou v atributu `description` zobrazeny informace o provedené transakci.

Tato kolekce bude též vyhledávána pomocí `userId`, proto i zde bude nutné vytvořit vyhledávací index nad tímto atributem. Při zpracovávání příchozích webhooků z platební brány Stripe bude nutné pro uložení dat o aktuálním spojení uživatele vyhledávat informace o členství uživatele. Z tohoto důvodu je též vhodné přidat vyhledávací index i k atributu `stripeCustomerId`.

3.7 Architektura

Návrh architektury je zobrazen na obrázku 3.5. Vychází z původního návrhu architektury, jejíž návrh provedl a implementoval Ing. Vít Štefan. Tato část, tedy komunikace jednotlivých komponent nástroje Timer2Ticket mezi sebou (na obrázku fialově) a spolu se službami třetích stran pro synchronizaci (na obrázku červeně) zůstala beze změn. S částí nového frontendu pracuje uživatel, který zde provádí úkony. Ta pak zasílá prostřednictvím REST požadavky do části T2T API, které jsou zde zpracovány. T2T Core se pak stará o synchronizaci dle nakonfigurovaného spojení. S touto částí komunikuje T2T API, který zasílá REST požadavky na okamžitou synchronizaci nebo informuje tuto komponentu o změnách v konfiguraci, vytvoření nebo úpravě spojení či o změně aktivity spojení.

Obě backendové části komunikují se službami pro synchronizaci, a to taktéž pomocí REST API. Část Core se službami komunikuje za účelem provádění synchronizačních jobů, část API z důvodu získávání dat nutných při konfiguraci spojení.



Obrázek 3.5 Návrh architektury nástroje Timer2Ticket, vytvořeno pomocí [74], bez notace

Prvním rozšířením je přidání autentizační platformy Auth0, se kterou bude komunikovat jak nový frontend, tak i T2T API. Nový frontend bude s Auth0 komunikovat s využitím knihovny na přímo pro přihlášení a získání přístupového tokenu, který bude následně součástí požadavků zasílaných do části API. V této části pak dojde taktéž s využitím poskytnuté knihovny k autentizaci volání.

Ostatní změny se týkají pouze komerční verze nástroje. Oddělení částí kódu pro nový frontend a část API již bylo popsáno v podkapitole Oddělení komerční a nekomerční části. Zbývá tedy popsat nově přidané komponenty.

Komunikace s platební bránou Stripe bude probíhat pomocí oficiální knihovny. Část API bude zasílat požadavky pro získání odkazu platební relace nebo zobrazení zákaznického portálu. O provedených změnách bude následně část T2T API informována prostřednictvím webhooků zasílaných na určený koncový bod pomocí REST požadavků. Příchozí události budou obsahovat informace o uskutečněných úkonech uživatele, které budou po zpracování aktualizovány v databázi.

Druhou přidanou komponentou je nástroj Fakturoid. Jelikož je však třeba do této komponenty pouze vkládat faktury vytvořené v platební bráně Stripe, komunikace probíhá pouze jednosměrně, s využitím REST API. Převod faktur bude probíhat tak, že po obdržení webhooku o zaplacení faktury z platební brány Stripe bude vytvořen požadavek pro vytvoření totožné faktury v nástroji Fakturoid.

3.8 Případy užití

Pro nefunkční prototyp nového frontendu již byly případy užití vydefinovány v bakalářské práci Bc. Jakuba Čermáka [15]. Vzhledem k faktu, že tato práce na vytvořený prototyp navazuje, není nutné vytvářet nové případy užití, ty tak budou převzaty z této bakalářské práce. Jelikož však došlo k částečné změně architektury, především ve spojitosti s využíváním platební brány Stripe, je nutné některé případy užití upravit. Níže tedy budou popsány úpravy pro případy užití, které neodpovídají nové implementaci nástroje Timer2Ticket.

UC2: Přiřazení členství uživateli

Vzhledem k rozsáhlým změnám bude pro větší přehlednost tento upravený případ užití popsán znovu. Do značné míry je však inspirován původním popisem [15].

Po registraci nebo při neúspěšné opakované platbě se uživatel dostane do aplikace, avšak jelikož je bez členství, nemůže využívat služeb nástroje. V rámci tohoto případu užití bude uživateli zobrazena nabídka členství včetně možnosti přikoupit spojení nad rámec členství. Po zvolení členství a případné úhrady poplatku mu bude moci umožněno využívání služeb nástroje Timer2Ticket.

Aktéři: Přihlášený uživatel bez zvoleného členství.

Počáteční podmínky: Nezaregistrovaný uživatel byl po registraci přesměrován do aplikace nebo se do aplikace opětovně přihlásil uživatel, který po registraci nezvolil žádné z nabízených členství a aplikaci opustil. Popřípadě se nezdařila opakovaná platba za členství uživatele, který se nyní do aplikace přihlásil.

Základní scénář:

1. Uživatel z nabídky zvolí chtěné placené členství.
2. Uživatel klikna na tlačítko „Zaplatit“.
3. Uživatel bude přesměrován na stránku platební brány, kde bude zobrazen formulář s platebními a fakturačními údaji.
4. Uživatel vyplní formulář.
5. Uživatel klikne na tlačítko „Odebírat“.

6. Systém zkontroluje, že byl formulář vyplněn korektně.
7. Systém informuje uživatele o úspěšném zaplacení členství. Uživatel následně bude přesměrován zpět do aplikace na stránku „Spojení“.

Alternativní scénář: Uživatel v prvním kroku zvolí členství zdarma, tedy Hobby s žádným přikoupeným spojením nad rámec členství. Následně klikne na tlačítko „Pokračovat“. V tomto případě bude též přesměrován na stránku platební brány, avšak bude zobrazen pouze formulář s fakturačními údaji. Následně scénář pokračuje 4. krokem. V 7. kroku je uživatel informován o korektním vyplnění formuláře a uživatel je zpět přesměrován do aplikace na stránku „Spojení“.

Exception scénář: Tento scénář je spuštěn ve chvíli, kdy je zjištěno nesprávné vyplnění formuláře v 6. kroku základního scénáře. Systém v tomto případě informuje o chybějících údajích a uživatel pokračuje 4. krokem základního scénáře.

UC7: Zobrazení všech spojení

V tomto případě užití byla provedena změna pouze ve zobrazování částečně nakonfigurovaných spojení. Ty budou zobrazeny pouze pokud uživatel neopustí webovou stránku. Částečně nakonfigurovaná spojení totiž nebudou ukládány trvale.

UC27: Změna hesla

Zde se scénář liší v tom, že pro změnu hesla bude uživatel přesměrován na webovou stránku platformy Auth0 pro změnu hesla. Základní scénář tedy bude začínat tím, že uživatel nejprve stiskne tlačítko „Změnit heslo“, čímž dojde k přesměrování na webovou stránku s formulářem pro změnu hesla. Po přesměrování bude uživatel pokračovat bodem 1. původního scénáře. Po dokončení původního scénáře dojde ke zpětnému přesměrování na stránku „Nastavení“.

UC28: Zobrazení emailové adresy

Tento případ užití zůstane zachován v případě, že má již zvolenou e-mailovou adresu. Ta bude zvolena, pokud se přihlásí způsobem, který umožňuje získání e-mailové adresy uživatele. V případě, že při přihlášení není možné e-mailovou adresu získat, uživatel bude moci na stejném místě zadat svou e-mailovou adresu. Po zvolení e-mailové adresy zůstane zachováno původní chování, tedy že si ji uživatel nebude moci sám změnit.

UC29: Zobrazení emailové adresy

V tomto případě užití bude vynechána možnost nastavení notifikací informujících o obdržených okamžitých synchronizacích. Změněn bude též výchozí stav, kdy zbývající notifikace bude zapnutá.

UC31: Aplikace kupónu

Jelikož z důvodu nízké priority nebude podpora kupónů implementována, tento případ užití může být odstraněn.

UC32: Nastavení platebních údajů

Největší změnou tohoto případu užití je pro nastavení platebních údajů využití platební brány Stripe. Na stránce platby tedy uživatel bude muset kliknout na tlačítko „Změnit platební údaje“, čímž bude přesměrován na zákaznický portál platební brány Stripe. Zde bude moci upravit své platební nebo fakturační údaje kliknutím na příslušné tlačítko a poté vyplnění formuláře.

UC33: Zobrazení přehledu všech zpracovaných objednávek

Podobně jako v předchozím případu užití, i tento je realizován prostřednictvím zákaznického portálu. Na něj se též dostane kliknutím na tlačítko „Zobrazit zpracované objednávky“, kdy si uživatel v dolní části stránky bude moci zobrazit všechny zpracované objednávky, včetně možnosti zobrazení a stažení faktury.

Návrh stránky členství

Pro tuto stránku se autor rozhodl nedělit obsah na jednotlivé případy užití. Proto ani zde nebudou jednotlivé případy užití rozváděny. Bude však změněno rozložení tak, že na této stránce nebude box s platebními údaji. Jeho funkci nahradí zákaznický portál platební brány Stripe, viz UC32.

Dále bude odděleno zakoupení členství a okamžitých synchronizací. Pro změnu členství bude uživatel též přesměrován na zákaznický portál platební brány Stripe, a to po kliknutí na tlačítko „Změnit členství". Na samotné stránce Členství bude zobrazen pouze ceník nabízených členství. Po přesměrování si bude moci vybrat členství i počet spojení nad rámec členství. Následně zadá platební a fakturační údaje.

Podobně bude realizován nákup okamžitých synchronizací. Na stránce Členství si zvolí počet. Při zadání nenulového množství okamžitých synchronizací bude uživateli zobrazena cena za zvolený počet (bez daně, ta bude dopočítána dle fakturačních údajů zákazníka na stránce platební brány). Po kliknutí na tlačítko „Koupit okamžité synchronizace" bude následně přesměrován na stránku platební brány Stripe pro zaplacení. Po uhrazení částky bude přesměrován zpět na stránku Členství.

Vzhledem k požadavku „P16 – Logování použitých okamžitých synchronizací" bude nutné v případě komerční verze rozšířit stránku Logy o možnost zobrazení použitých okamžitých synchronizací. Z tohoto důvodu budou přidány dva případy užití. Pro oddělení od původního číslování případů užití bylo změněno číslování přidaných případů užití tak, že byla změněna předpona na „UCx".

UCx1: Zobrazení logů okamžitých synchronizací

V rámci tohoto případu užití si uživatel bude moci zobrazit logy použitých okamžitých synchronizací. Ty budou zobrazeny formou tabulky, podobně jako logy jobů. Z tohoto důvodu bude rozdělena stránka „Logy" na dvě podsekce, podobně jako stránka „Profil". První podsekcí bude „Logy jobů", druhou „Logy okamžitých synchronizací". Přepínat mezi sekcemi bude možné v horizontálním menu. Tímto bude částečně upraven případ užití UC24: Zobrazení všech logů.

Tabulka s logy použitých okamžitých synchronizací tedy bude zobrazena po přechodu do podsekce „Logy okamžitých synchronizací". V tabulce budou obsaženy nejen použité okamžité synchronizace, ale i jejich zakoupení. Bude stránkovaná a bude obsahovat celkem pět sloupců:

Datum V tomto údaji se nachází datum a čas použití okamžité synchronizace nebo nabytí okamžitých synchronizací

Typ Údaj značí, zdali byla okamžitá synchronizace použita na synchronizace konfiguračních objektů nebo časových záznamů, popřípadě došlo-li ke koupi okamžitých synchronizací.

Změna počtu V tomto sloupci bude označen rozdíl v počtu okamžitých synchronizací oproti předchozímu stavu.

Nový zůstatek Tento sloupec bude zobrazovat nový zůstatek počtu okamžitých synchronizací po provedení změny.

Popis V případě použití okamžité synchronizace zde budou obsaženy informace o spojení, v případě nákupu informace o ceně a počtu zakoupených okamžitých synchronizací.

Řazení záznamů bude umožněno dle každého sloupce.

Aktéři: Přihlášený uživatel se zvoleným členstvím.

Počáteční podmínky: Přihlášený uživatel se zvoleným členstvím se nachází na podstránce „Logy okamžitých synchronizací" stránky „Logy".

UCx2: Zobrazení vybraných logů okamžitých synchronizací

Tímto případem užití je uživateli umožněno zobrazit si omezené množství logů použitých okamžitých synchronizací, podobně jako UC25: Zobrazení vybraných logů pro logy jobů. V tabulce bude umožněno filtrovat dle tří sloupců: datum, typ a popis. Aplikované filtry bude možné zrušit pro každý sloupec zvlášť. V případě, že danému filtru neodpovídá žádný záznam, bude uživateli zrušeny všechny použité filtry jedním kliknutím.

Aktéři: Přihlášený uživatel se zvoleným členstvím.

Počáteční podmínky: Přihlášený uživatel se zvoleným členstvím se nachází na podstránce „Logy okamžitých synchronizací“ stránky „Logy“.

Kapitola 4
Implementace

Následující kapitola se zaměřuje na popis implementovaných změn a rozšíření. Zahrnuje také možné situace, které by se mohly v budoucnu vyskytnout. Mezi tyto situace patří například změna konfigurace služeb třetích stran, úprava parametrů nabízených členství, případně též přidání nové služby k synchronizaci.

4.1 Povýšení Toggl Track API

Prvním zásahem do zdrojového kódu nástroje v rámci této diplomové práce bylo povýšení Toggl Track API z verze v8 na v9. Tento úkon musel být proveden do konce března roku 2023, jelikož poté by již původní verze API nebyla přístupná, čímž by se integrace Timer2Ticket s Toggl Track stala nefunkční. Současně bylo povýšeno Toggl Track Reports API na verzi v3 z současné verze v2. Původní verze sice bude v provozu až do konce června 2023, avšak nabízí se současně povýšit i toto API.

Povýšení verzí API nástroje Toggl Track se týkala především změn URL adres příslušných endpointů a úpravy zpracování odpovědí požadavků. Například v původní verzi byla požadovaná data obsažena v elementu `data` v těle odpovědi, nyní se již nacházejí přímo v těle. Pro povýšení bylo využito oficiální dokumentaci API [26] s průvodcem migrace na danou verzi [75].

Nejvýraznější změna se udála v získávání časových záznamů. Toggl Track API v8 totiž vracelo záznamy dávkovaně, kdy na jeden dotaz bylo vráceno pouze 50 záznamů, ostatní byly dostupné s využitím stránkování. Tomu tak již není, v současné verzi jsou vráceny všechny časové záznamy najednou. Z toho důvodu tedy není nutné procházet veškeré časové záznamy opakovaným voláním příslušného endpointu s daným číslem stránky.

Další drobná změna je, že v případě získání časového záznamu podle ID daný endpoint vrací pouze ID tagů. Jelikož jsou v implementaci použity názvy tagů, nikoli jejich ID, je nutné ID namapovat na názvy. Z tohoto důvodu je při zpracování odpovědi odeslán dotaz na získání všech tagů, ze kterých se následně získá název dle ID tagu. Zde bohužel není jiná cesta, jak získat název tagu dle ID, Toggl Track API získání tagu dle ID nepodporuje.

4.2 Autentizace a autorizace

Pro zajištění autentizace a autorizace uživatelů bylo nutné upravit nový frontend a část API backendu nástroje Timer2Ticket. Následující podsekce popisuje provedené změny.

Výpis kódu 4.1 Vložení přístupového tokenu do hlavičky URL požadavku

```
const authService = getInstance();
const backendUrl = 'http://localhost:3001/api/v2/';
const accessToken = await authService.getTokenSilently();
const data = {
    'headers': {
        'Authorization': `Bearer ${accessToken}`,
    }
};

let responseData = null;
await axios.get(backendUrl, data)
    .then((response) => {
        responseData = response.data;
    })
    .catch((error) => {
        console.error("Error fetching user connections.");
    });
```

Výpis kódu 4.2 Autentizace uživatele při zaslání dotazu na daný endpoint

```
const checkJwt = auth({
    audience: Constants.authAudience,
    issuerBaseURL: Constants.authDomain,
});

router.post('/', checkJwt, async (req, res) => {
    ...
}
```

Nový frontend

V původní verzi byla v novém frontendu implementována pouze prostá autentizace uživatele, která zabránila nepřihlášenému uživateli navštívení sekcí pro přihlášené uživatele. Na straně frontendu tak kromě drobných úprav zbývalo vyřešit autentizované volání backendu. I tuto zbývající část však řeší návod, jenž použil Bc. Jakub Čermák při původní implementaci autentizace uživatele [76].

Nejprve se tedy do konfigurace Auth0 musela přidat hodnota `audience`, kterou lze nalézt ve webovém rozhraní platformy. Následně si již stačí vyžádat přístupový token zavoláním metody `getTokenSilently()`, která byla vytvořena dříve ve stejném návodu, a její hodnotu vložit jako „bearer credential“ do hlavičky `authorization`. Zjednodušená ukázka zaslání HTTP požadavku s přístupovým tokenem je obsažena ve výpisu kódu 4.1.

Backend

Na backendu, respektive v API části, kde probíhá synchronizace, bylo nutné nejprve vyřešit autentizaci požadavků. Platforma Auth0 však pro snadnou práci s autentizací poskytuje npm balíček `express-oauth2-jwt-bearer`, který validuje JWT přístupové tokeny. Práci s ním popisuje návod [77]. Stačí do zdrojového kódu zkopírovat hodnoty `audience` a `issuerBaseURL`. Poté již stačí vložit middleware `checkJwt` k endpointu, u nějž bude vyžadována autentizace uživatele, jako je tomu ve výpisu kódu 4.2.

Výpis kódu 4.3 Ukázka použití knihovny `node-auth0` k získání formuláře pro změnu hesla

```
const management = new ManagementClient({
    domain: Constants.authManagementDomain,
    clientId: Constants.authManagementClientId,
    clientSecret: Constants.authManagementClientSecret,
  });
const data = {
    result_url: `${req.get('origin')}/profile/settings`,
    user_id: req.params.auth0UserId,
};

let response;
try{
    response = await management.createPasswordChangeTicket(data);
} catch (ex) {
    ...
}
```

Z přístupového tokenu lze však vyčíst některé informace, jako například ID uživatele v platformě Auth0. Toho lze využít při ověření, zdali zadané `auth0UserId` v URL požadavku je skutečně to, jež je obsaženo v přístupovém tokenu. To se děje i v části T2T API, jelikož všechny cesty, které vyžadují autentizaci, začínají `/api/v2/users/:auth0UserId`. Hodnota `auth0UserId` následně slouží i pro získání dat uživatele z databáze.

Přístupový token v části T2T API slouží dále k získání více informací o přihlášeném uživateli, konkrétně k získání e-mailu uživatele, je-li zadán. Z tohoto důvodu je volán koncový bod `GET /userinfo` Auth0 autentizačního API [78].

Auth0 Management API

Pro provádění administrativních úkonů nad platformou Auth0, jako je například smazání Auth0 účtu nebo změna hesla, je nutné použít Auth0 Management API, které je oddělené od Auth0 autentizačního API a poskytuje přístup ke správě a konfiguraci služby Auth0. Mimo výše zmíněného je možné pomocí tohoto API provést vše, co lze vykonat ve webovém rozhraní nástroje. [79]

Jelikož je vhodné mít obě API oddělené, je proto vhodné ve webovém prostředí Auth0 vytvořit ještě jednu aplikaci pro správu služby, a to typu „Machine to Machine“. Následně bude využito již existující systémové API „Auth0 Management API“. V sekci „Permissions“ lze zjistit, že toto API má všechny práva ke správě. Posledním krokem pak je autorizace aplikace v sekci „Machine to Machine Applications“. Zde je potřeba zvolit aplikaci vytvořenou z prvního odstavce textu a následně vybrat požadovaná oprávnění. V případě Timer2Ticket stačí zvolit pouze `create:user_tickets`, který umožní generovat URL adresy pro změnu hesla uživatelů.

V části T2T API je pro komunikaci s Auth0 Management API využita knihovna `node-auth0`. Nejprve je nutné iniciovat třídu `ManagementClient`, při níž je nutné zadat `domain`, `clientId` a `clientSecret` z vytvořené „Machine to Machine“ Auth0 aplikace. Následně již stačí zavolat příslušnou metodu s parametry a volitelným zpětným voláním, jež je zavoláno v případě chyby.

Ukázka kódu 4.3 zobrazuje inicializaci třídy `ManagementClient` a vygenerování URL pro změnu hesla uživatele. V této metodě je nutné zadat URL, na kterou bude uživatel po změně hesla přesměrován a ID uživatele v platformě Auth0. V těle odpovědi je pak obsažena URL, na které si uživatel může změnit heslo.

Výpis kódu 4.4 Ukázka načtení knihovny v souboru `main.js`, pokud se jedná o komerční verzi aplikace

```
if (Constants.IS_COMMERCIAL_VERSION) {
    console.info("Using commercial version");
    import('timer2ticket-client-vue-library').then((module) => {
        Vue.use(module.default);
    });
}
```

4.3 Oddělení komerční a nekomerční části

Oddělení komerční a nekomerční části bylo realizováno vytvořením dvou knihoven, jedna pro nový frontend a druhá, backendová, pro část T2T API. Detailnější popis obou vytvořených knihoven lze nalézt v následujících samostatných sekcích.

Je-li použita nekomerční verze aplikace, nejsou pro uživatele nástroje aplikovány žádné omezení. To znamená, že uživatel může mít libovolné množství spojení, synchronizaci je možné provádět každou hodinu a omezeny nejsou ani okamžité synchronizace.

4.3.1 Implementace frontendové knihovny

Pro část nového frontendu byla vytvořena knihovna komponent, při jejíž tvorbě bylo postupováno dle návodu [80], avšak místy byl postup lehce upraven. Návod popisuje samotnou inicializaci knihovny, její konfiguraci a adresářovou strukturu. Knihovna tak obsahuje veškeré komponenty zobrazující informace související s komerční verzí. Obsahují tak především komponenty pro podsekce členství a platby v sekci profil, ale lze v nich najít i komponenty určené pro sekci spojení nebo kontextové menu uživatele. Dále jsou v knihovně obsaženy funkce, které jsou potřeba v základní open-source verzi frontendu, avšak do knihovny nemohla být přesunuta celá komponenta. Typicky jsou to například funkce pro synchronizaci uživatelských údajů o členství nebo funkce, které umožňují zvalidovat konfiguraci plánu synchronizace spojení dle členství uživatele. Knihovna tak rozšiřuje základní open-source projekt frontendové části nástroje Timer2Ticket o podporu komerční verze.

Jelikož má být knihovna neveřejná, je nutné vyřešit její distribuci pouze lokálně, bez jejího zveřejnění. Npm, správce balíčků pro JavaScript, takové chování podporuje příkazem `npm link`. Vyvoláním tohoto příkazu v kořenovém adresáři knihovny vytvoří symbolický odkaz na tuto knihovnu. Pro použití knihovny v projektu je poté nutné přejít do kořenové složky projektu, v němž chceme knihovnu použít, a zadat příkaz `npm link package-name`, čímž je vytvořen symbolický odkaz na knihovnu ve složce `node_modules`. [81]

Takováto instalace knihovny je však vhodná pro testování, kdy je na ní nutné ještě pracovat a není tak nutné opakovaně knihovnu sestavovat. Po provedení implementace by bylo vhodné publikovat soukromý balíček do npm. K takovému balíčku poté mají přístup pouze konkrétní uživatelé nebo organizace. Konfigurace přístupu k balíčku proběhne při zadání příkazu `npm init`. Detailněji je publikování soukromých balíčků popsáno v [82].

Před použitím knihovny je však nutné provést její import. Pro komponenty se v souboru `main.js` pomocí klíčového slova `import`, avšak pouze v případě, jedná-li se o komerční verzi aplikace. Proto je import obalen do podmínky a použit ve funkci `then`, která je zavolána, pokud je import úspěšně proveden. Výše uvedené je vyobrazeno v ukázce kódu 4.4. Poté již lze komponenty používat ve zdrojovém kódu, v případě Timer2Ticket jsou použity taktéž podmíněně, pokud se jedná o komerční verzi. Část kódu `<t2t-membership-info v-if=isCommercialVersion />` tak zobrazuje použití komponenty `t2t-membership-info` z knihovny v komponentě z projektu v části `template`. [80, 83]

Výpis kódu 4.5 Ukázka načtení a použití backendové knihovny

```
let t2tLib: any;
if (Constants.isCommercialVersion) {
    t2tLib = require('timer2ticket-backend-library');
}

...

if (Constants.isCommercialVersion) {
    t2tLib.importedFunction();
}
```

Použití funkce z knihovny je přímočařejší, než použití knihovny. Poté lze provést import funkce z knihovny: `import {fetchMembershipData} from "timer2ticket-client-vue-library"`. Následně je možné naimportovanou metodu v projektu používat.

4.3.2 Implementace backendové knihovny

Pro backend, respektive část T2T API, byla vytvořena knihovna dle návodu [84]. Knihovna je implementována v jazyce TypeScript a její využití je podobné jako v případě knihovny pro nový frontend. V knihovně jsou obsaženy objekty a funkce, které taktéž souvisí s komerční verzí. Pro přehlednost a zachování jednoduchosti ve zdrojovém kódu však některé části určené pro komerční verzi zůstaly v projektu části T2T API. Jedná se především o definici cest vystaveného API a volání databáze včetně souvisejících objektů.

Import knihovny je totožný jako v případě frontendové knihovny, tedy pomocí příkazu `npm link`. Jelikož je implementace provedena v TypeScriptu, je nutné nejprve zavolat příkaz `npm run build`, který přeloží zdrojové kódy do JavaScriptu, včetně provedených změn. [84]

Pro načtení knihovny ve zdrojovém kódu byla využita funkce `require`, která je použita i ve výše zmíněném návodě [84]. Ta má navíc tu výhodu, že je možné ji využít podmíněně, v případě Timer2Ticket pouze v případě komerční verze aplikace. Ukázkové načtení knihovny pro backend a její použití je zobrazeno v ukázce kódu 4.5.

4.4 Napojení platební brány

V této podkapitole bude popsáno napojení platební brány Stripe. Pro komunikaci byla využita oficiální knihovna pro Node.js. Pro autentizaci je nutné použít API klíč, který lze získat ve Stripe dashboardu. Následně je již možné začít využívat metody poskytnuté knihovnou, jejichž popis si lze přečíst v oficiální dokumentaci [49].

Vytvoření předplatného

Po vstupu do aplikace je uživatel, který nemá zvolené členství, vyzván k jeho zvolení. Zároveň si může přikoupit i spojení nad rámec členství. Po potvrzení se odešle požadavek na část T2T API, která volbu uživatele zpracuje.

Při zpracování dojde ke kontrole, zdali uživatel již nemá členství. Má-li členství, požadavek nebude zpracován. Dále je zjištěno, zdali je již vytvořena instance zákazníka v platební bráně Stripe. To je zjištěno tak, že má uživatel vyplněno `stripeCustomerId` v dokumentu `MembershipInfo` náležejícímu uživateli. Zákazník v platební bráně Stripe je vytvořen zavoláním knihovní metody `stripe.customers.create`, kdy je to atributu `description` vloženo ID uživatele v databázi. ID

uživatele se předává pro případné snazší spárování s daty v databázi při problémech s platební branou. Následně je ID vytvořeného zákazníka v platební bráně stripe uloženo v atributu `stripeCustomerId`.

Je-li požadavek validní, je následně vygenerována URL adresa odkazující na platební bránu Stripe, kam bude uživatel přesměrován pro vyplnění platebních údajů. K tomu slouží metoda `createNewSubscriptionSession` ve třídě `StripeCommons` v knihovně pro backend. Ta vytvoří požadavek s využitím knihovní metody `checkout.sessions.create`, která vytvoří platební relaci. Vytvářet relaci pro provedení platby je doporučený postup [49].

Parametry této metody jsou informace, které produkty uživatel kupuje a jejich množství. Položka je reprezentována ID ceny položky. Z tohoto důvodu jsou při konfiguraci vyžadovány ID cen nabízených produktů. Jelikož se jedná o předplatné, je nutné nastavit parametr `mode` na `subscription`. Dále je při volání metody určen zákazník pomocí jeho ID v platební bráně Stripe.

Pro správný výpočet daně je nutné nastavit v parametru `automatic_tax` hodnotu atributu `enabled` na `true`. Tím dojde k automatickému výpočtu daně dle fakturačních údajů uživatele. Posledními podstatnými parametry je zvolení `billing_address_collection` na `required` a `payment_method_collection` na `if_required`. Tím je zajištěno, že zadání platební metody bude vyžadováno pouze tehdy, je-li cena nenulová (tedy nebudou vyžadovány při zvolení členství zdarma), avšak vždy bude vyžadováno zadání fakturačních údajů. Fakturační údaje jsou tak vyžadovány i při volbě členství zdarma, a to z důvodu případného zakoupení placeného členství, kdy je nutné dle fakturačních údajů určit daň.

Při implementaci bylo nutné předejít situaci, kdy by si uživatel zakoupil více členství, například otevřením aplikace ve více oknech a následnému zakoupení několika členství najednou. To je ošetřeno s využitím atributu `stripeLastSubscriptionSessionId`, ve kterém je uloženo ID poslední platební relace uživatele. Atribut je při vytváření nové platební relace kontrolován a je-li vyplněn, je relace uložená v tomto atributu zrušena pomocí zavolání knihovní metody `stripe.checkout.sessions.expire`. Teprve poté se vytvoří relace nová. Tímto je zabráněno zakoupit si více členství, jelikož při pokusu úhrady původní (již zrušené) relace vrátí platební brána chybu a platba není provedena. [49]

Zakoupení okamžitých synchronizací

Zakoupení okamžitých synchronizací je realizováno též pomocí platební relace. Odlišnost oproti zakoupení předplatného je v parametru `mode`, který má hodnotu `payment` značící jednorázovou platbu.

Zákaznický portál

Zákaznický portál platební brány Stripe je v implementaci využit pro změnu nebo zrušení členství uživatele, pro změnu platební metody a fakturačních údajů a pro zobrazení historie objednávek, respektive zobrazení či stažení faktur.

Na zákaznický portál se uživatel dostane kliknutím na tlačítka „Změnit platební údaje“ nebo „Zobrazit zpracované objednávky“ v podsekci platby sekce profil. I zde je odeslán požadavek na část T2T API, která pro získání adresy URL odkazující uživatele na zákaznický portál, využívá knihovnu. Použita je metoda `stripe.billingPortal.sessions.create`, které jsou v parametru předány ID uživatele v platební bráně Stripe a adresa URL, na kterou bude uživatel přesměrován po návratu ze zákaznického portálu.

Další možností přechodu na zákaznický portál je kliknutím na „Změnit členství“ v boxu „Změna členství“ nacházejícího se v podsekci členství sekce profil. Použitím této možnosti je však uživatel přesměrován přímo na výběr členství v zákaznickém portálu. To je dosaženo pomocí tzv. přímých odkazů, kdy lze voláním stejné metody určit, na jakou stránku zákaznického portálu bude uživatel přesměrován. Přesměrování přímo na výběr členství je tak dosaženo parametrem

`flow_data`, v jehož atributu `type` je zadáno `subscription_update`. Stripe nabízí celkem 4 možnosti přímých odkazů, které lze detailněji prostudovat v dokumentaci. [50]

Webhooky

O událostech, které jsou v platební bráně Stripe vykonány, se nástroj Timer2Ticket, respektive část T2T API, dozvídá pomocí webhooků. Nejsou však zpracovávány všechny typy zasílaných událostí, ale pouze čtyři.

Prvním zpracovávaným typem je `customer.subscription.updated`, který je zaslán po změně členství. Po zpracování jsou aktualizovány hodnoty o současném členství uživatele, kdy členství končí a počet aktivních spojení. Taktéž jsou upraveny plány synchronizace spojení, došlo-li ke snížení členství a je nutné plány upravit.

Dalším zpracovávaným typem událostí je `customer.subscription.deleted`, který informuje o zrušení členství. V případě nástroje Timer2Ticket může být tato událost zaslána pouze v případě, nezdaří-li se provést opakovanou platbu. I zde jsou aktualizovány hodnoty o členství uživatele, jeho konci a atribut `stripeSubscriptionId` na `null`, dále je maximální počet aktivních spojení stanoven na 0. Stejně tak musí být upraveny plány synchronizace spojení.

Třetím typem událostí, který je zpracováván, je `invoice.payment_succeeded`. Tato událost je vyvolána ve dvou případech – vytvoření předplatného nebo zakoupení okamžitých synchronizací. Rozeznány jsou tak, že v prvním případě je v atributu `billing_reason` obsažena hodnota `subscription_create`, druhý je pak identifikován pomocí ID ceny okamžitých synchronizací. Teprve poté je událost zpracována.

V případě vytvoření spojení jsou stejně jako v případě úpravy a smazání spojení aktualizovány hodnoty související s členstvím uživatele. Při zakoupení okamžitých synchronizací je zvýšena hodnota atributu `currentImmediateSyncs` v dokumentu `MembershipInfo` náležející danému uživateli o počet zakoupených synchronizací a současně vytvořen záznam v tabulce logů okamžitých synchronizací o zakoupení.

V rámci zpracování tohoto typu událostí ještě dochází k převodu faktur do nástroje Fakturoid. Tento úkon je však detailněji popsán v následující sekci textu.

Posledním typem události, který je zpracováván, je `invoice.marked_uncollectible`. Tato událost je zasílána, když se nepodaří zaplatit fakturu, k čemuž též dochází pouze pokud je neúspěšná opakovaná platba. Jelikož se v případě neúspěšné platby zruší předplatné, je potřeba zrušit vystavenou fakturu, aby nemohla být dodatečně uhrazena ze zákaznického portálu. Za tímto účelem tedy dochází ke zpracování této události. Pro zrušení faktury je využita metoda z oficiální knihovny platební brány Stripe `stripe.invoices.voidInvoice`. [49]

Příchozí webhooky jsou verifikovány. Tím je zajištěno, že budou zpracovávány pouze příchozí události pocházející z platební brány Stripe. Pro verifikaci je použita knihovní metoda `stripe.webhooks.constructEvent`, která bere jako parametry obsah hlavičky `stripe-signature`, nezpracované tělo požadavku a tajný klíč koncového bodu. Tajný klíč koncového bodu lze získat například při vytváření zasílání webhooků na daný endpoint ve Stripe dashboardu. Výše uvedené je implementováno v metodě `constructEvent` třídy `StripeCommons` v knihovně pro backend a je volána po obdržení každého webhooku.

Převod faktur do nástroje Fakturoid

Poslední oblastí, která souvisí s platební bránou Stripe, je převod zaplacených faktur do nástroje Fakturoid. Jak již bylo zmíněno, informace o zaplacené faktuře obdrží část T2T API pomocí Stripe webhooku.

Aby však bylo možné vytvořit fakturu v nástroji Fakturoid, je nutné nejprve vytvořit v tomto nástroji kontakt reprezentující uživatele v tomto nástroji. K tomu jsou využita data uvedená při platbě v platební bráně Stripe.

Kontakt je však vytvořen pouze v případě, nebyl-li vytvořen dříve. Tato informace je udržována v atributu `fakturoidSubjectId`. K vytvoření tedy dojde, je-li hodnota tohoto atributu prozatím `null`. Fakturoid poskytuje evidovat u kontaktu v atributu `custom_id` možnost vlastního identifikátoru. Tento atribut byl využit pro vložení ID zákazníka v platební bráně Stripe.

Pokud již je kontakt vytvořen, dojde k aktualizaci dat dle příchozích informací. Následně je možné vytvořit samotnou fakturu. Nejprve jsou převedeny všechny položky na faktuře z formátu platební brány Stripe do formátu nástroje Fakturoid. Získány jsou informace o názvu produktu, množství, dani a jednotkové ceny.

Vzhledem k tomu, že ve faktuře může být uplatněn zůstatek uživatele, který může vzniknout například předčasným zrušením předplatného, je ho nutné při převodu faktury zohlednit. Použitý zůstatek je vypočítán z atributů `starting_balance` a `ending_balance` a byl-li použit, je uveden i na převedené faktuře v nástroji Fakturoid mezi uhrazenými položkami. Údaje na faktuře jsou pak totožné.

Při vytvoření faktury je dále potřeba kromě jednotlivých položek vložit informace o kontaktu, jemuž faktura náleží a měnu faktury. I faktura poskytuje možnost evidovat vlastní identifikátor ve stejnojmenném atributu. Atribut je tedy využit pro vložení ID faktury v platební bráně Stripe.

Vytvořená faktura je však po vytvoření ve stavu vystavená. Jelikož je však již v platební bráně uhrazená, je potřeba ji označit jako zaplacenou (tedy dostat do stavu zaplacená). Provést tuto akci nelze při vytvoření, je proto potřeba vytvořit ještě jeden požadavek, který tuto akci vykoná. O to se stará metoda `markInvoiceAsPaid` ve třídě `FakturoidCommons` v knihovně pro backend. [65]

4.5 Změny provedené v novém frontendu

V novém frontendu bylo nutné upravit většinu stávajících komponent. Jejich grafické rozložení povětšinou zůstalo stejné, místy bylo nutné upravit určité grafické prvky, avšak vnitřní logika aplikace byla výrazně pozměněna. Nejvýznamnější změnou pak je, že je nový frontend již plně funkční, tedy že komunikuje s backendem nástroje Timer2Ticket a zobrazuje tak skutečná data uživatele získaná z databáze. Jednotlivé změny, úpravy a doplnění v novém frontendu jsou pak popsány níže v textu.

Načítání údajů

Po vstupu uživatele na web a jeho přihlášení jsou na pozadí načtena uživatelská data voláním příslušného endpointu backendu. Jedná-li se o komerční verzi, načtena jsou i data o členství uživatele. Načtení dat uživatele je realizováno v `navigationGuard` a děje se tak pouze pokud již data nebyla načtena a uložena s využitím pluginu Vuex, tedy pouze při prvním vstupu uživatele na stránku.

Směrování

Upraveno muselo být též směrování. Částečně bylo popsáno v předchozí části textu popisující načítání údajů uživatele. Směrování pomocí Vue Routeru však dále řeší, aby si v případě používání nekomerční verze aplikace nemohl zobrazit stránky, které jsou přístupné pouze v komerční verzi. Stejně tak je nutné uživateli neumožnit zobrazit si stránku `/withoutMembership`, tedy výběr členství, nemá-li zvolené členství, s již zvoleným členstvím. Má-li tedy uživatel již zvolené členství a pokusí se tuto stránku navštívit, bude přesměrován na stránku spojení.

Uživatel bez zvoleného členství

Přihlásí-li se do komerční verze aplikace bez zatím zvoleného členství, je přesměrován na stránku pro jeho zvolení. Tato stránka zůstala bez větších změn, pouze bylo přidáno pole pro možnost přidání spojení nad rámec členství. Následně bude přesměrován na platební bránu, kde v případě placeného členství uhradí částku za vybrané služby. Na platební bránu bude uživatel přesměrován i při výběru členství zdarma, a to z důvodu zadaní fakturačních údajů, které budou použity při případném přechodu na placené členství.

Drobná změna je taktéž v zadávání fakturačních údajů, které jsou nově zadávány na webové stránce platební brány Stripe oproti původní verzi, ve které byly zadávány na stránce nástroje Timer2Ticket.

Kontextové menu uživatele

Drobná změna se uskutečnila i v kontextovém menu uživatele, které lze zobrazit kliknutím na zelený kruh v pravém horním rohu obrazovky. Zde byl v jeho horní části zobrazen e-mail uživatele. Jelikož však vzhledem k možnému přihlášení přes sociální sítě nemusí být e-mail poskytnut, v případě jeho absence je zobrazeno jméno uživatele na dané sociální síti. Není-li k dispozici ani toto jméno, není zobrazen žádný údaj.

Sekce spojení

V sekci spojení nebyly provedeny žádné změny zobrazovaných grafických prvků, některé však byly doplněny, respektive přidány. Na první pohled největším rozdílem je pouze přidání prefixu „Nový“ u nově vytvořených spojení, které zatím nebyly uloženy. To je vytvořeno z důvodu, aby uživatel mohl snadno poznat, že spojení zatím není uloženo. Zároveň není možné ani zobrazit ID daného spojení, jelikož je ID přiřazeno vytvořenému spojení backendem až po jeho zpracavání.

V boxu konfigurace synchronizace bylo doplněno validování zadané konfigurace dané synchronizované služby. Validace probíhá odesíláním dat na backend. Data jsou odesílána průběžně během zadávání údajů uživatelem. Při čekání na odpověď jsou u daných polí zobrazena načítací kolečka, které informují uživatele o zpracovávání údajů. V případě validních dat je poté zobrazena zelená fajfka, v případě chyby červený křížek s popisem chyby pod příslušným polem. Po stisknutí tlačítka „Uložit“ je též zobrazeno načítací kolečko, po zpracování požadavku je pak zobrazena notifikace, zdali byla operace úspěšná či nikoliv.

Notifikace informující o provedení či neúspěchu operace byly přidány i u tlačítek okamžitých synchronizací a smazání spojení. Tyto notifikace jsou zobrazeny i při změně plánu synchronizace, která v případě již vytvořeného spojení probíhá na pozadí aplikace. Není proto třeba změnu potvrzovat samostatným tlačítkem.

Taktéž bylo doplněno u zobrazovaných časových údajů zobrazení časové zóny, detailněji je však tato změna popsaná v sekci Volba časové zóny.

Vzhledem k tomu, že boxy zobrazují informace o posledních synchronizacích, je vhodné uživatelům zobrazovat aktuální data, tedy především pokud právě probíhá synchronizace. Z tohoto důvodu byla využita technika „polling“, kterou též využil ve své implementaci Ing. Vít Štefan [12]. Každých 5 sekund je tak zaslán na backend dotaz pro získání aktuálních dat, které jsou následně frontendem zpracována a správně zobrazena. Uživatel tak nemusí stránku aktualizovat, aby viděl, zdali okamžitá synchronizace, která probíhá, již byla dokončena.

Sekce logy

Tato sekce též doznala řady změn. Nejvýraznější změnou je rozdělení sekce na dvě podsekce, podobně jako sekce profil. Důvodem tohoto rozdělení je nutnost zobrazování logů použití okamžitých synchronizací uživatele, což logicky náleží do této sekce. Horizontální menu tak obsahuje možnost zobrazení logů jobů a logů okamžitých synchronizací.

Podsekce logy jobů

Tato podsekce obsahuje původní obsah sekce logů, který však byl upraven, respektive doplněn. Úpravy zahrnují formátování zobrazovaných časových údajů, které jsou popsány v sekci Volba časové zóny. Dále byl upraven sloupec zobrazující status jobu. Filtrování tohoto sloupce bylo umožněno pouze na základě dvou stavů: „Ok“ a „Problematická“. Tyto stavy byly přejmenovány a doplněny o možnost filtrování probíhajících jobů. Nyní je tedy možné filtrovat dle tří možností: „Úspěšný“, „Neúspěšný“ a „Probíhá“.

U jednotlivých řádků (tedy logů) je stav jobu zobrazen pomocí tří ikon. Byl-li job úspěšný, je zobrazena zelená fajfka, v případě neúspěchu pak žlutý trojúhelník s vykřičníkem. Pokud

zatím job nedoběhl, a je tedy naplánovaný nebo již probíhá jeho zpracovávání, zobrazí se načítací kolečko. Změnou zobrazovaných ikon byl mimo jiné částečně vyřešen požadavek P44.

Aby byly zobrazované údaje vždy aktuální, je podobně jako v sekci spojení využita technika „polling“. Uživatel tak nemusí manuálně obnovovat stránku pro zobrazení aktuálních dat. V tabulce bude vždy zobrazeno maximálně sto záznamů, které budou maximálně 90 dnů staré. Toto chování vychází z implementace diplomové práce Ing. Víta Štefana [12].

Podsekce logy okamžitých synchronizací

Podsekce logy okamžitých synchronizací je velmi podobná podsekci logy jobů, liší se však v zobrazovaných údajích, jak napovídá název podsekcí. Odlišné jsou však i jednotlivé sloupce.

Pro logy použitých okamžitých synchronizací bylo vyhodnoceno, že je dostatečné zobrazit pouze pět sloupců. První zobrazuje datum použití okamžité synchronizace a druhý její typ. Typ může být trojího druhu: „Nákup“, přikoupil-li si uživatel okamžité synchronizace, a dva druhy pro použití okamžité synchronizace dle typu synchronizace: „Použití - časové záznamy“ a „Použití - konfigurační objekty“.

Následující dva sloupce zobrazují změnu počtu okamžitých synchronizací a jejich nový zůstatek, aby měl uživatel větší přehled o jejich použití. Posledním sloupcem je pak popis záznamu, ve kterém bude obsaženo buďto informace o spojení nebo informace o provedené koupi okamžitých spojení.

Tato podsekce též využívá techniky „polling“ pro zobrazení aktuálních dat. Taktéž však budou zobrazovaná data omezena na posledních 100 záznamů. Záznamy se však nebudou po 90 dnech mazat, jako tomu je u logů jobů.

Sekce profil

Jelikož se tato sekce skládá ze tří podsekcí, budou tyto podsekce popsány jednotlivě. Dvě z podsekcí, členství a platby, jsou však zobrazeny pouze v případě komerční verze aplikace.

Podsekce nastavení

Z této podsekce byla vyjmuta část pro zadávání kupónů, protože nebude v této práci implementována. Dále byl zprovozněn výběr časové zóny, kde však byla změněna nabídka výběru časových zón, též popsaná v sekci Volba časové zóny.

Z části notifikací bylo odstraněna možnost notifikace o obdržených okamžitých synchronizacích dle požadavku P34. Jelikož je správa přihlašování delegována na platformu Auth0, byl též odstraněn formulář pro změnu hesla. Ten byl nahrazen tlačítkem, které uživatele přesměruje na web Auth0 pro změnu hesla. Tlačítko je však aktivní pouze v případě, že uživatel pro přihlášení nevyužívá sociální síť. V opačném případě pozbývá tato akce smysl. Upravena byla též možnost zadávání e-mailové adresy dle požadavku P6.

Podsekce členství

V podsekci členství byl odstraněn box pro zadání platebních údajů. Ty budou spravovány v platební bráně Stripe a případnou změnu bude možné provést z podsekce platby. Též bylo odstraněno tlačítko pro zrušení předplatného, taktéž z důvodu delegování tohoto úkolu na zákaznický portál platební brány Stripe, a tlačítko pro prohlédnutí si souhrnu, jelikož při platbě bude uživatel přesměrován na stránku platební brány.

Po výše uvedených změnách tak v této podsekci zbyly pouze dva boxy. Box pro změnu členství obsahuje pouze informace o nabízených členstvích, pro provedení změny je nutné kliknout na tlačítko, které uživatele přesměruje na stránku platební brány Stripe, kde bude moci změnu provést. Box pro okamžité synchronizace se téměř nezměnil, pouze přibylo tlačítko, kterým se uživatel dostane na stránku platební brány pro úhradu objednávky.

Podsekce platby

Tato podsekce původně obsahovala box pro zadání nebo změnu platebních údajů a přehled

zpracovaných objednávek. Jelikož obě akce bude uživatel nově vykonávat přes zákaznický portál, nachází se zde pouze tlačítka, které uživatele přesměrují právě na zákaznický portál platební brány Stripe.

4.6 Změny provedené v backendu

Implementace backendu probíhala ve dvou částech. Nejprve byla upravena, respektive rozšířena část T2T API tak, aby podporovala veškeré operace prováděné na novém frontendu. Teprve poté byla upravena část T2T Core, která byla především přizpůsobena novým datovým strukturám a přidána podpora pro více spojení jednoho uživatele.

4.6.1 Změny v T2T Core

Tato část nedoznala tak rozsáhlých změn, jako je tomu u části T2T API. Přesto však došlo k několika významným změnám, které budou popsány níže.

Nový objektový model

Nejvýraznější změnou je změna modelu tak, aby odpovídala nově navrženému databázovému schématu. Kvůli této změně pak musel být upraven zdrojový kód části T2T Core na mnoha místech.

Změny provedené ve zdrojovém kódu však souvisí především se změnou třídy nesoucí definici synchronizovaného nástroje a drobnými změnami jejich atributů. Změnou je i to, že definice synchronizovaných nástrojů již nejsou uloženy v jednom atributu obsahující pole definic, ale v dokumentu spojení obsahující atribut explicitně pro definici prvního a druhého synchronizovaného nástroje. Poslední významnou změnou je pak to, že konfigurace spojení již není obsažena v objektu uživatele, ale v samostatném objektu pro každé spojení.

Kvůli výše uvedeným změnám došlo k úpravám na mnoha místech zdrojového kódu, především pak ve třídách pro synchronizační joby a třídách pro komunikaci se synchronizovanou službou. Samotná logika synchronizace časových a konfiguračních objektů změněna nebyla, synchronizace tedy probíhá stejně, jak tomu bylo i v původní verzi nástroje Timer2Ticket.

Se změnou datového modelu byla logicky změněna i třída `DatabaseService` sloužící pro komunikaci s databází. Dále byly provedeny drobné změny, jako například aktualizace stavu a času provedení posledního jobu v definici synchronizačních jobů či zohlednění časové zóny při plánování synchronizačních jobů.

Změna API

V souvislosti se změnou návrhu celého nástroje Timer2Ticket došlo i k úpravě API části T2T Core, které je voláno pouze z části T2T API. To nyní obsahuje pouze 4 koncové body, které jsou popsány níže v textu. Stejně jako T2T API, i zde došlo k přejmenování URL koncových bodů tak, že adresa obsahuje informace o verzi API.

První dva endpointy pro provedení okamžité synchronizace konfiguračních objektů a časových záznamů zůstaly zachovány. V cestě URL dotazu však již není obsaženo ID uživatele, ale ID logu jobu, které odkazuje na dokument uložený v databázi obsahující ID spojení, u něhož má být provedena okamžitá synchronizace.

ID logu jobu se zasílá z důvodu nutnosti logování použití okamžité synchronizace v komerční verzi. Záznam použití okamžité synchronizace musí být uživateli zobrazen hned po jeho použití. Jelikož však obsahuje i ID logu jobu, musí být vytvořeny společně. Z tohoto důvodu již není log jobu vytvářen v části T2T Core, ale už v T2T API, a následně je ID logu jobu zasláno. Zachování vytváření logu jobu a popřípadě i logu použití okamžité synchronizace v části T2T Core nebyla možná, jelikož je společná pro jak komerční, tak i nekomerční část.

Původní koncové body pro plánování, respektive zastavení všech jobů pro konkrétního uživatele, byly nahrazeny jedním koncovým bodem s novou cestou `/api/v2/update/`. Do těla požadavku pak je nutné do atributu `connectionIds` vložit poli ID spojení, které mají být aktualizovány.

Jak plyne z předchozího odstavce, tento endpoint bude volat T2T API, dojde-li ke změně či smazání spojení, nebo bude-li změněna časová zóna uživatele (z důvodu provádění synchronizace s ohledem na zvolenou časovou zónu). Tento endpoint bude volán i v případě změny, respektive snížení členství uživatele. V tomto případě je nutné volání, jelikož se při snížení členství mohou nějaká spojení deaktivovat, popřípadě změnit jejich plán synchronizace.

Tento koncový bod má za cíl zastavit všechny naplánované cron úkoly, aktualizovat plán synchronizace a aktivitu dle dat uložených v databázi a v případě, že je spojení aktivní, znovu naplánovat synchronizaci dle aktualizovaných hodnot.

Velice podobné chování má i koncový bod pro vytvoření spojení, které obsahuje ID nově vytvořeného spojení v URL cestě. Tento koncový bod navíc okamžitě provede synchronizaci konfiguračních objektů. Změnou chování je to, že tomuto se děje jen při vytvoření nového spojení, nikoliv při každé úpravě spojení.

Úklidové joby

Ke stávajícímu úklidovému jobu mazající logy jobů starší 90 dnů, který se spouští jednou za měsíc, byl přidán ještě jeden. Ten má za úkol mazat spojení označené ke smazání. Tato informace je obsažena v atributu `deleteTimestamp`. Spojení je smazáno nejdříve po dvou dnech od označení, nebylo-li uživatelem obnoveno. Se spojením jsou smazány i TESO náležející tomuto spojení. Tento přidaný job se spouští ve stejnou dobu, jako stávající úklidový job mazající logy jobů.

4.6.2 Změny v T2T API

Zásahy do této části byly patrně nejvýraznější, jelikož byl změněn či rozšířen objektový model, API aplikace i komunikace s databází. Provedené změny jsou tématicky rozděleny do několika textových sekcí.

Navržené API

V rámci změn došlo k vytvoření nového API, které podporuje všechny funkcionality prototypu nového frontendu. Stejně jako v původní implementaci, i zde bylo zachováno oddělení metod zpracovávajících REST požadavky do logických celků umístěných v samostatných souborech. Bylo tak vytvořeno 8 souborů, které jsou jednotlivě popsané v následujícím seznamu. Z následujícího výčtu jsou však poslední čtyři pouze pro komerční verzi aplikace.

Uživatelé

Tento logický celek obsahuje metody pro práci s objektem uživatele. První metoda slouží pro získání dat o uživateli. Zde je pomocí ID z platformy Auth0 dohledán příslušný záznam v databázi. Neexistuje-li záznam, jedná se o první přihlášení daného uživatele a je nutné záznam nejprve vytvořit. Pokud se navíc jedná o komerční verzi aplikace, je vytvořen též objekt `MembershipInfo` obsahující informace o členství uživatele. Metoda též obsahuje část pro spárování se zmigrovanými daty z původní verze nástroje Timer2Ticket. Tomu se však věnuje samostatná podkapitola Zpětná kompatibilita.

Následuje metoda `PATCH` pro úpravu atributů, které je možné měnit. Těmi jsou e-mailová adresa, nastavení zasílaných notifikací a časová zóna. Je-li daný atribut obsažen v těle požadavku a je validní, dojde k jeho úpravě. E-mailovou adresu je však umožněno změnit, respektive zvolit pouze v případě, že zatím nebyla zvolena. Přeje-li si uživatel změnit svou e-mailovou adresu, je nutné kontaktovat poskytovatele služby. Toto chování bylo zachováno z předchozí implementace.

Poslední metoda slouží pro změnu hesla uživatele. Metoda vygeneruje adresu, na kterou je následně uživatel přesměrován. Tím se dostane na stránku platformy Auth0, kde si může změnit svoje heslo.

Spojení

Metody v tomto logickém celku pracují s objektem spojení. Obsahuje metody pro získání všech spojení uživatele a získání jednoho uživatelova spojení dle ID spojení. Uživateli jsou však vrácena pouze nesmazaná spojení, tedy ty, kde není vyplněn atribut `deleteTimestamp`.

Následují metody `POST` a `PUT` pro vytvoření a úpravu spojení. Upravit spojení je možné i pomocí metody `PATCH`, pomocí které je možné podobně jako u uživatele upravovat jen ty atributy, které jsou obsaženy v těle požadavku. Takto je umožněno změnit aktivitu spojení a plány synchronizace.

Následuje metoda pro smazání a obnovení spojení. Obnovením spojení se rozumí smazání spojení, které se ale uživatel rozhodne vzít zpět. Posledními metodami je pak provedení okamžité synchronizace konfiguračních objektů nebo časových záznamů v daném spojení.

Logy jobů

Zde je obsažena pouze jedna metoda, a to pro získání logů jobů uživatele. Vráceno je však maximálně 100 nejnovějších záznamů.

Konfigurace synchronizovaných služeb

Metody nacházející se v tomto logickém celku zasílají dotazy na synchronizační služby. V tomto souboru došlo pouze k drobným změnám, původní logika metod zůstala zachována.

Členství

Tento logický celek je první, který je určen pouze pro komerční verzi. Nachází se zde pouze jedna metoda, která vrací informace o členství uživatele.

Logy použitých okamžitých synchronizací

Podobně jako u logů jobů, i v tomto případě se zde nachází pouze metoda pro získání logů použitých okamžitých synchronizací uživatele. I zde se vrací maximálně 100 nejnovějších záznamů.

Stripe

První ze dvou logických celků pro platební bránu Stripe. Obsahuje tři metody. První slouží pro získání odkazu k zaplacení zvoleného členství, druhá pro zaplacení jednorázového nákupu okamžitých synchronizací a poslední vrací odkaz pro vstup do zákaznického portálu platební brány. Detailněji jsou výše uvedené metody popsány v podkapitole Napojení platební brány.

Stripe webhooky

Druhý celek pro platební bránu Stripe slouží ke zpracovávání příchozích webhooků. Oddělení metod pro Stripe bylo provedeno z důvodu, že tento logický celek nevyžaduje autentizaci a autorizaci uživatele. I metoda pro zpracování Stripe webhooků je podrobněji popsaná v podkapitole Napojení platební brány.

Ve všech metodách, kde je to nutné, probíhá autentizace uživatele. Taktéž je kontrolováno, zdali má k datům, které požaduje, přístup. V tomto případě je kontrolováno, zdali ID uživatele v platformě Auth0 obsažené v přístupovém tokenu odpovídá s ID umístěném v URL požadavku.

Na začátku každého souboru je obsažen kód, který je společný pro všechny vytvořené REST metody. Zde u logických celků Členství, Logy použitých okamžitých synchronizací, Stripe a Stripe webhooky probíhá navíc kontrola, zdali se jedná o komerční verzi aplikace. Pokud ano, vykoná se příslušná metoda, v opačném případě je vrácen HTTP stavový kód 404 Not Found.

Autentizace a validace požadavků

S výjimkou endpointu pro Stripe webhooky vyžadují všechny metody autentizaci, tedy platný přístupový token, který uživatel získá přihlášením přes platformu Auth0. Validace vstupu od uživatele je pak prováděna pomocí knihovny class-validator [20]. Tato knihovna již byla využita v implementaci Ing. Víta Štefana [12], v jehož práci je popsáno, jak je validace s využitím této knihovny realizována.

Kontrola plánu synchronizací

Při vytváření nebo během úprav spojení je nutné ověřit, zdali je daný plán konfigurace validní s ohledem na členství uživatele. Za tímto účelem byla ve třídě `ValidateMembership` vytvořena metoda `validateSyncJobDefinition` knihovny pro backend. V této metodě je tedy ověřováno, zdali je v souladu se zvoleným členstvím umožněno vykonávat synchronizaci každou hodinu nebo více dnů v týdnu.

Neodpovídá-li plán konfigurace členství uživatele, je uživatel o této skutečnosti informován HTTP stavovým kódem a příslušnou chybovou hláškou. Samotná změna není realizována, popřípadě spojení není vytvořeno.

Aktivace a deaktivace spojení

V případě aktivace spojení je potřeba ověřit, zdali má uživatel dostatečný počet spojení, které může mít aktivní. O to se stará atribut `currentActiveConnections` třídy `MembershipInfo`. K její inkrementaci však dojde jen v případě, že je nižší než `currentConnections`, tedy maximální počet aktivních spojení uživatele.

Jelikož však může dojít k souběhu více operací nad tímto atributem, což by mohlo vést k nekonzistenci dat, je potřeba provést tuto operaci atomicky. Z tohoto důvodu byla při ověřování výše popsané podmínky a případné následné zvýšení hodnoty atributu v databázi MongoDB použita atomická operace `findOneAndUpdate` [85].

Při deaktivaci spojení je též snížení hodnoty provedeno atomicky použitím databázové operace `findOneAndUpdate`. Není však již použita podmínka ověřující dostupnost aktivních spojení, jelikož v tomto případě její aplikace postrádá smysl.

Použití okamžité synchronizace

Použití okamžitých synchronizací probíhá podobně, jako aktivace a deaktivace spojení. Počet dostupných okamžitých synchronizací uživatele je uložen v atributu `currentImmediateSyncs`, který se taktéž nachází ve třídě `MembershipInfo`. Změna hodnoty atributu též probíhá atomicky pomocí `findOneAndUpdate`. V případě použití okamžité synchronizace je navíc kontrolován počet, který musí být větší než 0, jinak se operace neprovede.

Změna členství

Je-li změněno členství, je potřeba zkontrolovat, zdali plán synchronizace a aktivita spojení stále odpovídá plánu, které nové členství nabízí. Sníží-li si uživatel své členství, je tak potřeba snížit i četnost synchronizace. Podobně v případě snížení maximálního počtu aktivních spojení je nutné některé spojení deaktivovat.

O toto se stará funkce `reconfigureConnections` v souboru `stripe_webhook.ts`, která je vyvolána vždy při změně členství v platební bráně Stripe nebo při zrušení členství. Parametry této funkce je původní objekt `MembershipInfo` uživatele, název nového členství a nový maximální počet aktivních spojení. Na základě těchto parametrů je provedena změna v příslušných spojeních uživatele.

Nejprve je určen počet spojení, které je potřeba deaktivovat. Následně je daný počet spojení deaktivován. Spojení jsou deaktivována od nejstaršího dle data vytvoření. Z tohoto důvodu bylo nutné oproti návrhu rozšířit třídu `Connection` o atribut `createdTimestamp` nesoucí informaci o datu vytvoření spojení. Během toho je též o příslušný počet deaktivovaných spojení změněna hodnota atributu `currentActiveConnections` v objektu `MembershipInfo` uživatele.

Výpis kódu 4.6 Ukázka formátování data v komponentě `ConnectionBox`

```
getFormattedDate(unixTimestamp) {
    const date = new Date(unixTimestamp * 1000);
    return date.toLocaleString(
        getTimeFormatOptions(this.$store.state.user.timeZone)
    );
}
```

Bylo-li sníženo členství, je provedena úprava plánu synchronizace všech spojení uživatele tak, aby odpovídaly novému členství. To je realizováno tak, že je iterováno přes všechny spojení uživatele. V případě, že plán synchronizace neodpovídá členství uživatele, je buďto zrušena synchronizace každou hodinu, nebo omezena četnost synchronizace z původního výčtu dnů pouze na nejdřívější den v týdnu.

O provedených změnách je následně uživateli odeslán informační e-mail. Pro ten bylo využito stávajících tříd `email_service.ts` pro samotné zaslání e-mailu a `translate_service` pro sestavení těla e-mailu dle jazyku uživatele. V současné chvíli jsou však e-maily odesílány pouze v anglickém jazyce, jako tomu bylo v původní implementaci Ing. Víta Štefana [12].

4.7 Volba časové zóny

Jak již bylo zmíněno v požadavcích, uživatel si může zvolit časovou zónu, aby plány synchronizace odpovídaly času v místě, ve kterém se právě nachází. Výchozím nastavením časové zóny po prvním přihlášení je Praha, uživatel si však může časovou zónu změnit v sekci nastavení profilu. Zde si uživatel vybere příslušnou časovou zónu z nabídky dle regionů a měst. Nabídka je převzata z [86] a byla použita z důvodu snadného použití s objektem `Date`.

Ve frontendové části je pak časová zóna využita při vypisování časových informací spojených s plánem synchronizace. Prvním výskytem je komponenta `ConnectionBox`, v jejíž záhlaví je zobrazena informace o poslední provedené synchronizaci. Informace o času poslední synchronizace přijde z backendu ve formátu unixového času, nejprve je tedy vytvořena instance objektu `Date`. Následně je využita metoda `toLocaleString`, která převede objekt `Date` na řetězec formátovaný dle konfigurace. Konfigurace je přijímána jako první atribut této metody. Jelikož se tato metoda používá na více místech ve frontendové části, byla vytvořena funkce `getTimeFormatOptions` vracející vždy totožnou konfiguraci formátu data. Tím bude dosaženo totožného formátu zobrazování dat napříč aplikací. Tato metoda přijímá jeden parametr, a to časovou zónu uživatele, která je získána z proměnné pluginu Vuex. Výše popsané je zobrazeno v ukázce kódu 4.6, kdy je pro zobrazení data využita metoda `getFormattedDate`. Vyobrazené datum též zobrazuje i časovou zónu, aby nedošlo ke zmatení uživatele a ten si tak v případě potřeby časovou zónu mohl změnit.

Druhým výskytem je pak komponenta `SynchronizationSchedule`, která naopak v záhlaví zobrazuje datum následující synchronizace. Následující datum je zde získáno s využitím cron výrazu, který definuje plán synchronizace a knihovny `node-cron`, která umožňuje získat následující datum synchronizace díky metodě `nextDates`. Předtím je však nutné vytvořit instanci objektu `CronJob`, která bere jako parametr i časovou zónu.

Následně je již využit stejný postup jako v předchozím odstavci. Získání textového řetězce následujícího data synchronizace je zobrazeno v ukázce kódu 4.7.

Dalším místem, ve kterém se vypisují časové údaje dle časové zóny, je tabulka logů. V tabulce jsou vypisovány časové údaje stejně jako v komponentě `ConnectionBox`. Jelikož však musí být formátován i výběr data v záhlaví tabulky, které slouží pro filtrování obsahu, musela být přidána ještě jedna konfigurace formátu data zobrazující pouze datum. Tu lze získat voláním funkce `getDateFormatOptions`. Díky tomu je při výběru data zobrazené datum správně naformátováno a zároveň je tak umožněno filtrování.

Výpis kódu 4.7 Ukázka formátování následujícího data synchronizace s využitím knihovny `node-cron` v komponentě `SynchronizationSchedule`

```
    calculateNextTime(schedule) {
        const CronJob = require('cron').CronJob;
        const job = new CronJob(schedule, null, null, null,
                                this.$store.state.user.timeZone);
        const nextDate = job.nextDates(1)[0];

        return nextDate.toLocaleString(
            getTimeFormatOptions(this.$store.state.user.timeZone)
        );
    }
```

S využitím funkce `getDateFormatOptions` je též zobrazeno datum, kdy uživateli skončí současné členství, a to v kontextovém menu uživatele a v podsekci členství sekce nastavení. Datum je zde však zobrazeno pouze v případě, že využívá některý z placených členství (tedy všechny kromě členství hobby s jedním spojením).

V části T2T Core je pak časová zóna využita při plánování provádění synchronizací. Při implementaci však bylo zjištěno, že je využita stejnojmenná, avšak jiná knihovna `node-cron` [87], než na kterou odkazuje Ing. Vít Štefan ve své diplomové práci [12]. I tato knihovna však přímočaře podporuje časové zóny.

Při vytvoření plánu je tak načtena informace o časové zóně uživatele z databáze. Ta je poté vložena jako třetí atribut do metody `cron.schedule`, která pravidelně spouští funkci předanou v druhém argumentu dle cron výrazu zadaného prvním argumentem.

4.8 Přidání nové služby k synchronizaci

Tato podkapitola popisuje nutné úpravy v souvislosti s přidáním nové služby k synchronizaci. Zmíněny však budou jen potřebné změny v souvislosti s implementací, která byla vykonána v rámci této diplomové práce. To se tedy týká především částí T2T API a nového frontendu. Pro část T2T Core platí kroky popsané v diplomové práci Ing. Víta Štefana [12].

Ten mimo jiné ve své práci zmiňuje, že v souvislosti s přidáním nové služby může být nutné rozšíření objektu `ServiceConfig` určující konfiguraci služby o další možné atributy, které nová služba potřebuje. Taktéž je zde zmíněno, že nelze vyloučit takovou odlišnost při přidávání nové služby, že bude třeba provést výraznější zásah do zdrojových kódů nějaké části. Při návrhu a následné implementaci provedené v této diplomové práci však bylo též dbáno na obecnost a jednoduchost přidání nové služby.

T2T Core

Kromě změn uvedených v diplomové práci Ing. Víta Štefana [12] je zapotřebí upravit třídu `Connection`. Tato třída nyní předpokládá existenci dvou služeb, tedy Redmine a Toggl Track. Při přidání další služby však bude nutné upravit logiku v této třídě, jelikož nástroj Timer2Ticket bude muset podporovat synchronizaci mezi službami stejného typu.

T2T API

V této části je nejprve nutné přidat definici nástroje v proměnné `ToolType`, která se nachází v souboru `type_of_tool.ts`. Definice zahrnuje atribut `name` pro jméno služby k synchronizaci. Následuje atribut `type` pro typ služby, tedy zdali se jedná o projektový nástroj nebo nástroj pro

měření času. Tento atribut zatím není v současné implementaci použit, byl však přidán s ohledem na budoucí rozšíření nástroje, kdy by mohl být využit pro určení typu spojení.

Posledním atributem je pak `attributesFromClient` obsahující povinné atributy pro konfiguraci dané služby. Před zpracováním daného požadavku je tak zkontrolováno, zdali požadavek obsahuje tyto povinné atributy. Validace probíhá ve třídě `ConnectionFromClient`, ve které je nutné provést úpravy, především vložení nové metody pro validaci přidávané služby.

Posledním bodem je pak úprava konstruktoru třídy `ServiceConfig`. Tato třída nese informace o konfiguraci služby, je proto třeba v konstruktoru z dat obdržených v požadavku správně vytvořit instanci této třídy.

Nový frontend

V části nového frontendu je taktéž nutné přidat definici nástroje ve stejnojmenné proměnné nacházející se v souboru `typeOfTool.js`. Poté je nutné novou službu k synchronizaci přidat do metody `getLabelToBoxHeading` v komponentě `ConnectionBox.vue`, která slouží k zobrazení informace o synchronizované službě v záhlaví boxu spojení, a do metody `getService` v souboru `shared.js` sloužící k mapování objektu obdrženého z backendu do objektu používaného v implementaci nového frontendu.

Poslední, avšak nejnáročnější věcí, je úprava komponenty `ToolSetUper.vue`. V této komponentě se nachází veškerá logika spojená s konfigurací služby k synchronizaci. V této komponentě je tedy potřeba vytvořit formulář pro konfiguraci nové služby, získávání dat ze služby, je-li to potřeba, validaci vstupu od uživatele a následné vložení dat do objektu, který bude odeslán v požadavku na frontend, rozhodne-li se uživatel uložit vytvořené spojení, respektive provedené změny v již existujícím spojení.

4.9 Změna ceny okamžitých synchronizací

Změna ceny okamžitých synchronizací zahrnuje změnu ceny balíčku nebo počet okamžitých synchronizací v balíčku. Začít je třeba ve Stripe dashboardu. Zde je nutné u produktu okamžitých synchronizací vytvořit novou cenu s požadovanými parametry a původní archivovat. Po vytvoření ceny je nutné vložit její ID namísto ID původní ceny v konfigurační proměnné `STRIPE_IMMEDIATE_SYNCS_PRICE_ID`. Tato proměnná je používána v knihovně pro backend.

Ve stejném projektu je poté nutné upravit množství okamžitých synchronizací v jednom balíčku v objektu `ImmediateSyncs` obsahujícím atribut `quantityInPackage`. Tento atribut slouží pro validaci uživatelova požadavku, aby si mohl přikoupit pouze takový počet okamžitých synchronizací, který je násobkem množství v balíčku.

Poté již zbývá upravit frontend aplikace. Jelikož se změna týká komerční verze, je nutné zdrojové kódy upravit taktéž v knihovně pro frontend. Jediný výskyt informace o ceně a počtu okamžitých synchronizací v balíčku v boxu okamžitá spojení v podsekci členství sekce profil.

Jelikož však možná změna těchto parametrů byla zohledněna při implementaci s důrazem na co nejsnazší úpravu, stačí pouze změnit hodnoty atributů objektu `IMMEDIATE_SYNC` umístěného v souboru `typeOfMembership.js`. Tento objekt obsahuje dva atributy, jeden pro cenu za balíček a druhý pro množství okamžitých spojení obsažených v balíčku.

4.10 Změna nabízených členství

Pod tímto pojmem si lze představit změnu ceny nabízeného členství nebo aktivního spojení nad rámec členství, změnu četnosti synchronizace či změnu počtu aktivních spojení v ceně členství. I zde je nutné začít vytvořením nové ceny ve Stripe dashboardu u produktu, jehož cenu chceme změnit. Také je nezbytné změnit popisek produktu, není-li již aktuální. Při vytváření ceny je nutné ohlídat, aby všechna nabízená členství měla stejnou cenu za spojení nad rámec

členství, jelikož takovýto byl návrh v bakalářské práci Bc. Jakuba Čermáka [15], podle kterého probíhala implementace.

Ve frontendové části stačí též změnit pouze příslušné atributy v souboru `typeOfMembership.js`. Soubor se nachází v knihovně určené pro nový frontend. Zde je konstantní objekt `MembershipType` obsahující tři objekty s totožnou strukturou, každý pro jedno nabízené členství. Počet spojení v ceně členství určuje atribut `connectionNumber`, cenu stanovuje `price`.

Četnost synchronizace definují atributy `isSyncEveryDayEnabled` a `isSyncEveryHourEnabled`. Současná implementace pracuje se třemi možnými četnostmi synchronizace – každý týden, den nebo hodinu, které lze dosáhnout kombinací výše uvedených atributů. Bude-li v budoucnu potřeba přidat jinou možnou četnost synchronizace, bude nutné provést další změny v implementaci.

Posledním krokem v knihovně pro frontend je změna ceny pro aktivní spojení nad rámec členství. Tuto změnu je nutné provést v atributu `price` objektu `CONNECTIONS_OVER_MEMBERSHIP`. Ten se nachází ve stejném souboru, jako objekt definující parametry nabízených členství.

Nutné je také provést změny v části knihovny pro backend. Ty jsou obdobné provedeným změnám ve frontendové knihovně. Zde se taktéž nachází soubor `typeOfMembership.ts` obsahující objekt `MembershipType` s totožnou strukturou. Zde tedy stačí jen změnit počet spojení v ceně členství a atributy určující četnost synchronizace. Dále je třeba nezapomenout na nastavení proměnných prostředí nesoucí ID ceny daného členství, které jsou obsaženy v souboru `.env`.

4.11 Konfigurace částí nástroje Timer2Ticket

Již v implementaci Ing. Víta Štefana byly jednotlivé části konfigurovány prostřednictvím proměnných prostředí. Takto byly například konfigurovány URL adresy T2T API a T2T Core, přístup k databázi nebo e-mailový klient.

Pro přehlednost a snazší vývoj byly v projektech všech částí nástroje Timer2Ticket vytvořeny konfigurační soubory `.env` v kořenovém adresáři projektu. Zde jsou definovány všechny proměnné prostředí, které využívá Timer2Ticket při úvodní konfiguraci. Pro načtení proměnných z tohoto souboru pak byla využita knihovna dotenv [88].

Změna konfigurace platformy Auth0

Při integraci autentizační a autorizační platformy Auth0 bylo bráno v potaz i možnost snadné záměny uživatelských účtů z této platformy. S přihlédnutím k vydání projektu jakožto open-source tak bude muset každý, kdo bude chtít Timer2Ticket nasadit na vlastní server, vložit vlastní konfiguraci Auth0 aplikace.

Pro konfiguraci bude třeba upravit pouze proměnné prostředí v částech T2T API a v novém frontendu, tedy v částech, kde se autentizace uživatelů využívá. Nejprve je však nutné vytvořit v platformě Auth0 aplikace a API tak, jak je popsané v podkapitole Autentizace a autorizace.

V projektu nového frontendu je nutné změnit tři konfigurační proměnné v souboru `.env` s předponou `VUE_APP_AUTH0_`. Hodnoty těchto proměnných jsou též popsány v podkapitole Autentizace a autorizace a lze je získat ve webovém rozhraní nástroje Auth0.

V projektu části T2T API jsou konfigurační proměnné taktéž obsaženy v souboru `.env`. Jelikož je v tomto projektu použito jak autentizační, tak i Management API, je nutné nakonfigurovat obě. Pro konfiguraci autentizačního API jsou určeny proměnné `AUTH0_AUDIENCE` a `AUTH0_DOMAIN`, konfigurace Management pak probíhá pomocí tří proměnných `AUTH0_MANAGEMENT_CLIENT_SECRET`, `AUTH0_MANAGEMENT_CLIENT_ID` a `AUTH0_MANAGEMENT_DOMAIN`. Význam i těchto proměnných je detailněji popsán v podkapitole Autentizace a autorizace.

Změna konfigurace platební brány Stripe

Platební brána Stripe je též konfigurována prostřednictvím proměnných prostředí. Ukázka proměnných, které je třeba nakonfigurovat pro správné fungování platební brány Stripe, je v souboru `.env` knihovny pro backend. To proto, že jsou tyto konfigurační proměnné použity pouze v části T2T API, jedná-li se o komerční verzi.

V tomto souboru lze nalézt několik proměnných. Nejpodstatnější je však `STRIPE_SECRET_KEY` obsahující tajný API klíč. Následují proměnné obsahující ID cen položek členství a okamžitých spojení v platební bráně Stripe, které usnadňují práci při zpracování webhooků ve zdrojovém kódu. Předposlední proměnnou je pak URL adresa, na kterou je uživatel přesměrován po úspěšné platbě nebo po návratu ze zákaznického portálu platební brány.

Poslední proměnná `STRIPE_ENDPOINT_SECRET` nese tajný klíč koncového bodu, který je použit při verifikaci příchozích webhooků. Jeho hodnotu lze nalézt například při vytváření zasílání webhooků na daný endpoint ve Stripe dashboardu.

Změna konfigurace nástroje Fakturoid

Posledním nástrojem třetí strany použitým při implementaci je nástroj Fakturoid, který je též nutné konfigurovat. Ten vyžaduje zadání čtyř proměnných. Ukázka konfigurace je též obsažena v souboru `.env` knihovny pro backend.

První konfigurační proměnnou je `FAKTUROID_SLUG` a slouží jako identifikace účtu. Jedná se o název účtu, který je zadán při registraci. API však vyžaduje hodnotu zadanou malými písmeny. [65]

Proměnná `FAKTUROID_EMAIL` a `FAKTUROID_API_TOKEN` slouží pro autorizaci volání. První jmenovaná proměnná nese informaci o e-mailu zadaném při registraci, druhá obsahuje API token, který lze získat v nastavení uživatelského účtu. [65]

Poslední proměnnou je `FAKTUROID_T2T_ADMINISTRATOR_EMAIL`, který se zasílá v každém požadavku v hlavičce `User-Agent`. Tato hlavička slouží k identifikaci aplikace využívající Fakturoid API a e-mailová adresa obsažená v této proměnné obsahuje e-mailovou adresou na technického správce integrace.

Konfigurace komerční a nekomerční verze

Pro použití komerční verze nástroje Timer2Ticket je nutné nastavit proměnnou prostředí `IS_COMMERCIAL_VERSION` v části T2T API, respektive `VUE_APP_IS_COMMERCIAL_VERSION` v projektu nového frontendu na hodnotu `true`. V tomto případě budou použity neveřejné knihovny obsahující části kódu potřebné pro správné fungování obou částí aplikace. Nebudou-li knihovny k dispozici, aplikace fungovat nebude.

V případě, že se uživatel rozhodne použít open-source verzi, není nutné proměnné nastavovat, v tomto případě se použije výchozí hodnota `false`, která neveřejné knihovny nevyužívá.

4.12 Zpětná kompatibilita

Základní myšlenka dosažení zpětné kompatibility byla nastíněna v předchozí kapitole. Zde byl navržen jednosměrný migrační skript, který bude obsažen v samostatném projektu. Při implementaci skriptu byl využit popis provedených změn v databázovém modelu obsažený v podkapitole Databázový model spolu s novým databázovým schématem zobrazeným na obrázku 3.4.

Struktura nově vytvořeného projektu je poměrně jednoduchá, kromě nezbytných souborů pro konfiguraci samotného projektu se zde nachází starý a nový model objektů a služby pro komunikaci s databázemi. Ty jsou dvě – jedna pro databázi obsahující data z původní verze

nástroje a druhá nová, která bude využívána novou verzí. Chováním i možností konfigurace odpovídají třídě `DatabaseService` z částí T2T API a T2T Core, liší se však metodami.

Třída `InputDatabaseService` tedy načítá data z databáze pro původní verzi aplikace. Obsahuje pouze dvě metody, a to metodu pro získání všech uživatelů z databáze a metodu pro získání všech TESO uživatele. Zmigrovaná data jsou pak uložena do databáze s využitím třídy `OutputDatabaseService`, která obsahuje čtyři metody pro vytvoření příslušných dokumentů. Metodami lze do databáze vložit nového uživatele, spojení, TESO a informace o členství uživatele (dokument `MembershipInfo`).

Samotná logika migrace a podpůrné funkce jsou obsaženy v souboru `app.ts`. Zde jsou nejprve z původní databáze načteny veškeré objekty uživatelů, kteří následně budou migrováni. Migrace se však uskuteční jen pro uživatele, jejichž status je ve stavu `inactive` nebo `active`. Uživatelé ve stavu `registrated` nebudou migrováni, jelikož nemají vytvořenou konfiguraci služeb k synchronizaci a tudíž neexistuje spojení, které by šlo zmigrovat.

Následně se vytvoří nový objekt uživatele, který z původního objektu využije pouze e-mailovou adresu a datum registrace, pro ostatní atributy se použijí výchozí hodnoty. Objekt je následně uložen do databáze. Poté je vytvořen objekt `MembershipInfo` obsahují též výchozí hodnoty, jaké jsou nastaveny běžnému uživateli po prvním přihlášení do aplikace.

Dále je vytvořeno nové spojení, jehož konfigurace je obsažena v objektu uživatele ze staré verze nástroje Timer2Ticket. Konfigurace je tak převedena na nové datové objekty. Převeden však není plán synchronizace konfiguračních nebo časových objektů. Důvodem pro toto rozhodnutí je, že nově bude muset plán synchronizace odpovídat členství uživatele. Jelikož však členství bude zvoleno až po migraci, musí tedy odpovídat nejnižšímu členství z nabídky, tedy plánu hobby.

Plán synchronizace tedy bude nastaven tak, aby odpovídal synchronizaci jednou týdně. Den v týdnu a čas synchronizace bude zvolen náhodně. Po migraci je však předpokládáno, že si uživatel tento čas změní. Při změně plánu synchronizace tak bude moci zohlednit i jím vybrané členství.

Stav spojení bude taktéž změněn na neaktivní, a to bez ohledu na původní stav synchronizace. To bude z důvodu, aby nebyly zbytečně využívány výpočetní prostředky pro synchronizace spojení těch uživatelů, kteří například již nástroj Timer2Ticket nevyužívají nebo nechtějí využívat novou verzi. Očekává se tedy, že si uživatel po prvním přihlášení nejenže upraví plán synchronizace, ale též i aktivuje nově vytvořené spojení.

Následně již zbývá zmigrovat pouze TESO uživatele. Je tedy nutné jejich načtení z databáze pro původní verzi a následně jejich znovuvytvoření v databázi pro novou verzi nástroje. Objekty jsou převedeny téměř beze změny, jediný změněný atribut je `userId`, který byl nahrazen `connectionId` a odkazuje tak na ID spojení, jemuž náleží. Tento atribut tedy nese ID spojení, jež bylo též během migrace dat vytvořeno.

Logy jobů nebudou převedeny z původní verze nástroje na novou, a to z důvodu, že to není potřeba. Původní aplikace totiž nebyla placená a neplynou z ní jakékoliv závazky pro uživatele. Před přechodem na novou verzi budou uživatelé v dostatečném předstihu informováni o přechodu a o skutečnosti, že původní logy budou ztraceny.

V části T2T API je pak následně po prvním přihlášení při získávání dat o uživateli kontrolováno, zdali se v databázi nenachází zmigrovaný uživatel se stejnou e-mailovou adresou, který zatím nemá přiřazené žádné ID v platformě Auth0. Najde-li se takový uživatel, je přihlášený uživatel spárován se zmigrovanými daty a může tak využívat svou původní konfiguraci spojení.

4.13 Výsledek implementace

Z celkových 51 vydefinovaných požadavků v kapitole Analýza jich bylo implementováno 26. Jmenovitě se jedná o požadavky P1–P9, P11–P20, P22, P26, P28 a P31–P34. Další tři požadavky, P39 a P42–P43, není nutné vzhledem ke změně návrhu implementovat. Přehled všech požadavků spolu se stavem, ve kterém se nachází, je zobrazen v tabulkách 4.1 a 4.2. Rozdělení požadavků do dvou tabulek je provedeno z důvodu přehlednosti vzhledem k počtu řádků v tabulce.

Tabulka 4.1 Přehled požadavků se stavem realizace, první část

Číslo	Název požadavku	Priorita	Stav
Napojení nového frontendu na backend			
P1	Sjednocení autentizační a autorizační platformy	Must have	Realizován
P2	Podpora více spojení jednoho uživatele	Must have	Realizován
P3	Správa spojení	Must have	Realizován
P4	Možnost okamžité synchronizace spojení	Must have	Realizován
P5	Zobrazení logů	Must have	Realizován
P6	Nastavení e-mailu	Must have	Realizován
P7	Správa účtu	Must have	Realizován
P8	Volba časové zóny	Could have	Realizován
P9	Možnost volby notifikací	Could have	Realizován
P10	Zasílání notifikací	Could have	Otevřený
Zprovoznění monetizace			
P11	Výběr členství	Must have	Realizován
P12	Přikoupení aktivních spojení nad rámec členství	Must have	Realizován
P13	Podpora okamžitých synchronizací	Must have	Realizován
P14	Historie objednávek	Must have	Realizován
P15	Okamžité ukončení předplatného	Should have	Realizován
P16	Logování použitých okamžitých synchronizací	Must have	Realizován
P17	Napojení platební brány	Must have	Realizován
P18	Možnost stažení faktury	Should have	Realizován
P19	Převod faktur do nástroje Fakturoid	Could have	Realizován
P20	Uložení platebních údajů	Should have	Realizován
P21	Podpora kupónů	Could have	Otevřený
Oddělení komerční a nekomerční části			
P22	Oddělení částí zdrojových kódu	Must have	Realizován

Tabulka 4.2 Přehled požadavků se stavem realizace, druhá část

Číslo	Název požadavku	Priorita	Stav
Přidání nástroje Jira			
P23	Přidání nástroje Jira	Should have	Zanalyzován
P24	Podpora synchronizace časových záznamů mezi projektovými nástroji	Should have	Zanalyzován
P25	Přidání „enterprise“ verze synchronizace mezi projektovými nástroji	Should have	Zanalyzován
Zpětná kompatibilita			
P26	Migrace původní verze Timer2Ticket na novou verzi	Must have	Realizován
Údržba aplikace			
P27	Povýšení použitých knihoven na novější verze	Could have	Otevřený
P28	Povýšení Toggl Track API	Must have	Realizován
Zvýšení efektivity			
P29	Prevence obcházení okamžitých synchronizací	Could have	Otevřený
P30	Podpora webhooků	Won’t have	Zanalyzován
Dokončení nového frontendu			
P31	Přidání notifikací	Could have	Realizován
P32	Zobrazení probíhající okamžité synchronizace	Could have	Realizován
P33	Změny barvy a aktivity tlačítek, když uživatel může vykonat akci	Could have	Realizován
P34	Odstranění možnosti notifikací o obdržených okamžitých synchronizacích	Could have	Realizován
P35	Revize responsibilního zobrazení	Could have	Otevřený
P36	Úprava uživatelského rozhraní	Could have	Otevřený
P37	Úprava boční lišty	Could have	Otevřený
P38	Úprava boxů spojení	Could have	Otevřený
P39	Úprava formuláře s platebními údaji	Could have	Nerealizován
P40	Volba členství kliknutím na sloupec	Could have	Otevřený
P41	Předvýběr konkrétního členství při vstupu z propagačního webu	Could have	Otevřený
P42	Přidání frekvence placení k ceně v sumarizaci objednávky	Could have	Nerealizován
P43	Možnost filtrování objednávek	Could have	Nerealizován
P44	Úprava tabulky logů	Could have	Otevřený
P45	Přidání tutorialu	Won’t have	Otevřený
Propagační web			
P46	Dokončení všech sekcí propagačního webu	Could have	Otevřený
P47	Dokončení hlavičky propagačního webu	Could have	Otevřený
P48	Responzivní propagační web	Could have	Otevřený
P49	Přidání překladů	Could have	Otevřený
P50	Úvodní video	Could have	Otevřený
P51	Tutorial videa	Won’t have	Otevřený

Po dokončení implementační části diplomové práce došlo ve spolupráci s Maxem Hejdou, zaměstnancem Jagu s.r.o., k nasazení systému. Nejprve však bylo potřeba systém k nasazení připravit. Prvním krokem bylo publikování knihoven jakožto soukromých balíčků do GitHub npm registry [89]. Publikování se děje automaticky pokaždé, když dojde k vydání nové verze.

Poté následovaly další úkony. Pro projekt `timer2ticket-client-vue` bylo nutné vytvořit automatické vytváření Docker obrazů, taktéž s využitím GitHub Actions, podobně jako je tomu v backendových projektech. Následně byl Maxem Hejdou vytvořen konfigurační soubor Docker Compose [90], který umožňuje snadné spouštění vícekontejnerových aplikací v prostředí Dockeru.

Po diskuzi se zaměstnanci a spolupracovníky Jagu s.r.o bylo rozhodnuto, že nová verze nástroje Timer2Ticket bude nasazená na adrese `https://new.timer2ticket.com`. Současně bude zachována původní verze aplikace na adrese `https://app.timer2ticket.com` z důvodu interního testování. Teprve poté je v plánu migrace na novou verzi a odstavení původní verze. Po dokončení implementace této diplomové práce byla nová verze nástroje určena k internímu testování ve společnosti Jagu s.r.o., avšak je dostupná i veřejně.

Kapitola 5
Testování

V této kapitole budou popsány způsoby, jakými bylo implementované rozšíření nástroje Timer2Ticket v této práci otestováno.

5.1 Jednotkové testy

Základním typem testů, který testuje samostatné části kódu, jsou jednotkové (unit) testy. V části T2T API bylo navázáno na testy vytvořené v rámci diplomové práce Ing. Víta Štefana [12]. Z tohoto důvodu byl využit aplikační rámec Mocha [91], který byl použit v existujících jednotkových testech. Tento aplikační rámec byl doplněn frameworkem Chai, který je knihovnou pro ověřování tvrzení (*assertion library*). Chai byl tedy využit pro porovnávání hodnot pomocí poskytnutých metod [92]. Dále byl přidán framework Sinon.JS, který slouží pro vytváření falešných objektů, čímž je možno simulovat chování objektů dle testovacího scénáře [93]. V implementovaných testech byl použit v části T2T API pro simulování odpovědí při volání funkcí pro získání objektů z synchronizovaných služeb.

Pomocí jednotkových testů byly otestovány části T2T API a knihovny pro frontend i backend. Pro část T2T Core nebyly jednotkové testy implementovány. Jedním z důvodů je, že nebyla měněna logika aplikace, tato část byla pouze přizpůsobena na nové datové struktury. T2T Core však neobsahuje testy ani z původní implementace, jelikož by bylo nutné simulovat chování služeb třetích stran, popřípadě založení testovacích účtů u daných služeb a ty pak využít pro testování [12]. Ze stejného důvodu pak nejsou vytvořeny jednotkové testy ani v části T2T API.

5.1.1 Pokrytí kódu jednotkovými testy

Pokrytí kódu testy (též známé jako Code Coverage) je metrika určující, jaká část zdrojového kódu byla otestována pomocí testů. Vývojářům pomáhá zajistit, že je realizovaný software řádně otestován odhalováním oblastí, které testy postrádají. Pokrytí kódu bývá obvykle udáváno v procentech, které říká, jak velká část kódu je pokryta testy. [94]

Existují čtyři běžné typy pokrytí kódu, které stanovují, jakým způsobem dochází k výpočtu pokrytí: [94]

Function coverage Pokrytí funkcí je stanoveno procentem funkcí ve zdrojovém kódu, které jsou zavolané v testech.

Line coverage Pokrytí řádků měří, kolik řádků spustitelného kódu bylo testovací sadou vykonáno.

Tabulka 5.1 Přehled pokrytí kódu jednotkovými testy dle projektu

Projekt	Statements	Branches	Functions	Lines
timer2ticket-api	14.81%	8.02%	23.49%	15.50%
timer2ticket-backend-library	27.16%	40.00%	25.92%	25.30%
timer2ticket-client-vue-library	31.52%	60.00%	60.00%	31.52%

Branch coverage Pokrytí větví se zaměřuje na větve ve zdrojovém kódu a sleduje, které byly pokryty testy. Větví se v tomto případě rozumí například podmíněné výrazy (if/else) nebo smyčky.

Statement coverage Pokrytí instrukcí je určeno procentem počtu instrukcí ve zdrojovém kódu, které jsou testy provedeny. Metrika se na první pohled může zdát totožná s pokrytím řádků, avšak ta nebere v potaz řádky, které obsahují více instrukcí. Tyto metriky budou totožné, pokud bude jeden řádek vždy obsahovat jen jednu instrukci.

Tyto čtyři typy pokrytí kódu obsahuje většina nástrojů pro výpočet pokrytí, nevyjímaje tak ani ty, které byly použity v rámci této závěrečné práce.

Použité nástroje

Pokrytí kódu testy v části T2T API a knihovně pro backend bylo měřeno pomocí nástroje Istanbul [95], respektive s využitím klienta pro příkazový řádek nyc [96]. Istanbul navíc umožňuje měřit pokrytí i v jazyce TypeScript [97].

Nástroj jde velmi dobře konfigurovat pomocí argumentů při spouštění nástroje, popřípadě s využitím konfiguračních souborů. Konfigurace ve formátu JSON může být umístěna přímo v souboru `package.json` nebo v samostatném souboru. Použita byla druhá možnost, konfiguraci tak lze nalézt v souboru `.nycrc`. [96, 98]

V konfiguračním souboru byly vybrány soubory, pro které se má pokrytí kódu testy vypočítat (všechny soubory ve složce `src`) a typ výstupu nástroje, který je jak textový, tak i HTML. Dále jsou přidány atributy potřebné pro kompatibilitu s jazykem TypeScript.

V případě knihovny pro frontend bylo využito nástroje c8 [99]. Ten byl využit z důvodu, že při použití nástroje Istanbul v tomto projektu byly ve zprávě o pokrytí nesprávně uvedeny nulové hodnoty [100]. Nástroj c8 se však spouští a konfiguruje obdobně, jako výše zmíněný Istanbul, respektive jako klient pro příkazový řádek nyc [99].

Výsledné pokrytí kódu testy

Výsledky z měření pokrytí kódu testy jsou zobrazeny v tabulce 5.1, která obsahuje naměřené hodnoty pro projekt a typ pokrytí. Detailněji je způsob měření a komentář k výsledkům popsán níže v textu samostatně pro každý měřený projekt.

timer2ticket-api

Tento projekt dosahuje z testovaných projektů nejnižšího pokrytí kódu. Je tomu tak z důvodu zaměření jednotkových testů na složku `src/models` obsahující třídy používaných objektů. V tomto případě tedy byla testována správná logika vytváření objektů a následně správné chování metod. Většina souborů v této složce dosahuje pokrytí kódu 100 %.

Ostatní soubory, především pak soubory ve složce `src/routes` obsahující funkce zpracovávající REST požadavky uživatelů a ve složce `src/shared` obsahující metody například pro komunikaci s databází nebo e-mailovým klientem, jsou testovány jiným způsobem popsaným v následující podkapitole. Pro tyto soubory by, jak již bylo zmíněno, bylo nutné simulovat chování služeb třetích stran.

timer2ticket-backend-library
Přestože tento projekt má vyšší procento pokrytí kódu testy, taktéž ani zdaleka nedosahuje přijatelné hodnoty [101]. Jednotkové testy byly totiž zaměřeny především na funkce ve složce `src/enums`. Ostatní soubory buďto obsahují pouze definici tříd nebo metody pro komunikaci se službami třetích stran, které jsou též otestovány jiným způsobem.

timer2ticket-client-vue-library
Poslední testovací projekt dosáhl nejvyššího pokrytí kódu, které však bylo počítáno pouze ze souborů s příponou `.js`. Soubory, které nebyly otestovány, buďto neobsahují testovatelný kód, nebo slouží pro komunikaci s backendem nástroje.

5.2 Testování koncových bodů

V průběhu implementace a následně i po vytvoření nové verze nástroje Timer2Ticket byly testovány koncové body v částech T2T API a Core. Cílem těchto testů bylo vyzkoušet chování koncových bodů tak, jak je požadováno, včetně ověření zasílání správných stavových (návratových) HTTP kódů a obsahu těla odpovědí. Tímto způsobem je mimo jiné testována i komunikace se službami třetích stran. Jsou tak testovány i třídy a funkce, které z výše uvedených důvodů nebyly testovány jednotkovými testy. Testy v nástroji Postman tak jednotkové testy doplňují.

Tento druh testů lze považovat za systémové, respektive funkční testování, i přes absenci testovacích scénářů. Předpokládané chování lze však vyčíst ze seznamu požadavků a upravených případů užití. Těmito testy je však testováno, zdali se systém chová tak, jak by měl a je testován jako celek. Současně je testována i integrace mezi ostatními systémy, jako jsou například nástroje k synchronizaci nebo platební brána Stripe. [102, 103]

Pro tento účel byly vytvořeny dvě testovací sady v nástroji Postman [104]. Postman je platforma pro snadné testování a ladění HTTP API. Během implementace byly v tomto nástroji vytvořeny HTTP požadavky pro otestování všech koncových bodů, včetně ukázkových obsahů hlaviček, těl požadavků nebo dotazů v URL. Vytvořené sady byly přiloženy do zdrojových kódů obou částí a jsou obsaženy ve složce `test` v souboru `.postman_collection.json` s prefixem `T2T Core` nebo `T2T API` dle části, pro kterou je sada určena.

Ve stejné složce je dále přiložen soubor `.postman_globals.json` se stejnými prefixy pro nastavení proměnných v nástroji Postman. V souboru jsou obsaženy ukázkové hodnoty. Do proměnných je možné uložit hodnotu, kterou je možné následně využívat ve vytvořených požadavcích. Není tak nutné zadávat na více místech stejnou opakující se hodnotu [105]. Ve vytvořených testovacích požadavcích je pak proměnných využito například pro vložení přístupového tokenu, ID uživatele ve službě Auth0 nebo ID spojení, se kterým jsou prováděny operace. Díky proměnným je navíc umožněno pracovat s testovací sadou snadněji, jelikož například často se měnící přístupový token stačí změnit pouze na jednom místě.

Pro použití vytvořených testovacích sad v nástroji Postman je nutné výše zmíněné soubory před jejich použitím do nástroje importovat. Import se však provede snadno a intuitivně na hlavní obrazovce nástroje.

5.3 Testování na reálných datech

Podobně jako v diplomové práci Ing. Víta Štefana [12], i v této diplomové práci byla aplikace v průběhu implementace i po ní testována s použitím reálných dat. Oproti původní implementaci však bylo nutné otestovat nejen správné fungování synchronizovaných služeb, tedy Redmine a Toggl Track, ale také nástrojů a platforem, které byly integrovány. Zde se jedná o autentizační a autorizační platformu Auth0, platební bránu Stripe a nástroj Fakturoid.

Takto byla testována nekomerční i komerčně verze aplikace. V případě první jmenované se jednalo především o testy konfigurace spojení a následných úprav, včetně testování krajních hod-

not a nevalidních požadavků. Dále bylo vyzkoušeno správné zobrazování a chování logů a změna nastavení na stránce Profil. Taktéž byla otestována komunikace mezi části T2T API a Core. Chování části T2T Core bylo otestováno pomocí několika testovacích účtů v nástrojích Redmine a Toggl Track, kdy byla ověřená správná funkčnost synchronizace konfiguračních objektů a časových záznamů. V neposlední řadě bylo též otestováno validní chování všech částí i bez použití knihoven pro komerční verzi.

Podobně byla otestována i komerční verze aplikace. U této verze však ještě bylo otestována integrace s platební bránou Stripe a nástrojem Fakturoid. V případě platební brány byl otestován výběr, změna a zrušení členství, včetně případné změny plánu synchronizace a promítnutí změn do části T2T Core. Stejně tak byl otestován i nákup okamžitých synchronizací. V případě nástroje Fakturoid pak bylo ověřeno, že jsou faktury převedeny správně a obsahově odpovídají faktuře vytvořené platební bránou Stripe.

5.4 Kontrola naplnění požadavků se zadavatelem

Posledním typem testů, kterým byla implementace podstoupena, je kontrola naplnění požadavků se zadavatelem. Tento test byl realizován dne 9. listopadu 2023 po dokončení implementační části a nasazení aplikace. Vedoucí práce Ing. Jiří Hunka postupoval podobně, jak by postupoval nový uživatel využívající službu Timer2Ticket. Po vstupu na stránku se tedy nejprve zaregistroval, poté zvolil členství a následně si vytvořil nové spojení. Dále vstoupil na všechny sekce a vyzkoušel funkcionality, které nástroj nabízí. Během testování se vedoucí práce doptával na implementační detaily či na podrobnosti ohledně reprezentace dat v aplikacích třetích stran. Testování probíhalo hladce, avšak bylo odhaleno několik nedostatků, které budou samostatně popsány dále v textu.

První drobný nedostatek byl spatřen v zobrazování tabulky s nabízenými členstvími, včetně vlastností jednotlivých členství. Informace o vlastnostech jsou získávány ze souboru, který se však nachází ve zdrojových kódech frontendu i backendu. Tím dochází k duplicitě zdrojového kódu. V budoucnu by tak bylo vhodné získávat informace o členstvích z backendu pomocí REST dotazu, druhou možností je vytvoření knihovny obsahující části zdrojového kódu, které jsou použity ve více samostatných projektech nástroje.

Druhým shledaným nedostatkem bylo nemožnost vložení DIČ pro firemní zákazníky. Tento nedostatek byl však ihned po testování odstraněn rozšířením formulářů o možnost vložení DIČ. V případě vložení DIČ je též propsáno do zmigrované faktury v nástroji Fakturoid.

Vedoucí práce dále zmínil absenci uživatelského manuálu. S ním částečně souvisí požadavek „P51 – Tutorial videa“, popisující vytvoření videí, které uživatele blíže seznámí s nástrojem Timer2Ticket. Přesto by však bylo vhodné mít i textový uživatelský manuál, který bude obsahově velmi podobný těmto videím.

Patrně nejzávažnější nedostatek byl spatřen v absenci častější automatické synchronizace, například každých 10 minut. Tento nedostatek však nebyl zmíněn při sběru požadavků, tudíž nedošlo k jeho implementaci. Lze však očekávat, že v dohledné době k jeho implementaci dojde.

Další nedostatky se týkají nastavení profilu a spojení. Ing. Jiří Hunka by uvítal možnost změny e-mailu nebo zrušení účtu uživatelem. Vyžadovanou změnou je též přesunutí nastavení notifikačních e-mailů pro každé spojení zvlášť.

Následně byl diskutován formát faktur převedených do nástroje Fakturoid. Vedoucí práce zde požadoval, aby variabilní symbol faktury byl shodný s číslem faktury, což je však výchozí chování tohoto nástroje. Druhým požadavkem pak bylo, aby faktury ve Fakturoidu měly totožné číslo jako původní faktura z platební brány Stripe. Pro splnění tohoto musela být aplikace upravena.

Následně se Ing. Jiří Hunka zajímal o možnost využívání nasazené komerční aplikace zaměstnanci společnosti Jagu s.r.o. tak, aby nemusely být účtovány poplatky za placené členství. Pro tento účel bylo při zadávání platebních údajů do příslušného formuláře v platební bráně Stripe umožněno zadávat kupóny. V praxi tak může provozovatel vytvořit kupón s libovolnou výší slevy zadanou buď na fixní částku, nebo v procentech. Stripe navíc umožňuje široké nastavení, včetně

uplatnění slevy jen na určitý produkt či omezení pro konkrétního uživatele [50]. Lze předpokládat, že především omezení platnosti kupónu pro konkrétního uživatele bude využito, aby se předešlo zneužití kupónu pro poskytování placených služeb zdarma. Doplněním výše uvedeného tak byl částečně realizován požadavek „P21 – Podpora kupónů“. Realizace však neodpovídá původnímu návrhu z bakalářské práce Bc. Jakuba Čermáka.

Poslední tři nedostatky byly nalezeny v uživatelském prostředí aplikace. Tyto nedostatky však pochází z původní implementace nového frontendu a nevznikly při implementaci této diplomové práce. Jelikož však nebyly odhaleny při testování bakalářské práce Bc. Jakuba Čermáka, je zřejmé, že se jedná pouze o maličkosti. První výtkou bylo zobrazování notifikačních bublin, především těch chybových, kdy zadavatel vyžaduje jejich umístění v blízkosti místa, na kterém chyba vznikla. V současné implementaci jsou zobrazovány v pravém horním rohu obrazovky.

Druhou chybou v uživatelském prostředí aplikace bylo, že při volbě počtu okamžitých synchronizací, které si uživatel přeje zakoupit, nelze zadat hodnotu klávesnicí. Tento nedostatek byl opraven, v případě zadání nevalidního počtu okamžitých spojení byla přidána chybová hláška a uživateli je zablokováno tlačítko pro přesměrování na platební bránu.

Třetí vada vznikla pravděpodobně nepozorností, kdy se v české verzi aplikace při zobrazování četnosti synchronizace u členství Senior zobrazovala hodnota „jednou denně“ namísto „každou hodinu“. I zde došlo k nápravě.

Veškeré výše uvedené nedostatky, které nebyly po tomto testování opraveny či doimplementovány, jsou zaznamenány v nástroji Redmine.

5.5 Testování po nasazení

Po nasazení na adrese `https://new.timer2ticket.com` bude pokračovat interní testování v rámci společnosti Jagu s.r.o. Po dohodě se zadavatelem práce a budoucím provozovatelem služby již toto testování nebude řešeno v této diplomové práci. Lze však očekávat, že spolupráce na projektu Timer2Ticket bude nadále pokračovat především při případném opravování nalezených chyb a předávání znalostí o implementovaném systému.

Kapitola 6
Budoucí rozšíření

V této kapitole bude popsáno možné budoucí rozšíření a rozvoj nástroje Timer2Ticket. Budoucí směřování aplikace bude rozděleno do tří kategorií – první kategorií je rozšíření a úpravy navazující bezprostředně na tuto diplomovou práci. Následující dvě jsou určeny dle časového výhledu na krátkodobé a dlouhodobé cíle.

Pro všechny návrhy a rozšíření, které jsou dále v textu popsané, byly podobně jako v případě bakalářské práce Bc. Jakuba Čermáka [15] vytvořeny úkoly v projektovém nástroji Redmine. Úkoly byly vytvořené i pro požadavky z kapitoly Analýza, které v této práci nebyly realizovány. Současně byly aktualizovány stávající, již vytvořené úkoly tak, aby odpovídaly stavu po dokončení implementační části a testování v této diplomové práci.

6.1 Bezprostředně navazující rozšíření

V rámci rozšíření nástroje realizovaného v této diplomové práci by bylo vhodné upravit implementaci v několika oblastech. První oblastí je změna uložení proměnných obsahující citlivé informace. Těmi mohou být například přístupové klíče nebo API tokeny k integrovaným nástrojům, jako je platforma Auth0, nástroj Fakturoid nebo platební brána Stripe, popřípadě přístupové údaje do databáze. V implementaci Ing. Víta Štefana byly tyto informace vkládány pomocí proměnných prostředí.

Tato práce tento přístup zachovává, byl však přidán `.env` soubor obsahující tyto konfigurační proměnné. Takto jsou jednoduše odděleny citlivé údaje od samotného kódu a je takto dosažena snadná správa konfigurace aplikace. Pro snadný vývoj nástroje je však tento soubor umístěn v repozitáři, což lze považovat za bezpečnostní hrozbu. Proto by tento soubor bylo vhodné před nasazením do produkčního prostředí odstranit, čímž budou odebrány i citlivé údaje. To se však předpokládá, jelikož veškeré využívané nástroje třetích stran budou převedeny na účty provozovatele.

Druhou oblastí, kterou by bylo vhodné v dohledné době upravit, je zasílání notifikací e-mailem, což je vydefinováno v požadavku P10. Tento požadavek z důvodu nižší priority nebyl v této práci realizován, mimo jiné i kvůli probíhajícím paralelním úpravám logování chyb synchronizačních jobů ve zdrojovém kódu části T2T Core zaměstnanci společnosti Jagu s.r.o. Očekávané chování je takové, že bude-li mít uživatel nakonfigurováno zasílání e-mailů v případě problému při běhu synchronizačního jobu a zadanou validní e-mailovou adresu, bude o této skutečnosti informován právě zaslaným e-mailem.

Nelze opomenout ani na výstup z kontroly naplnění požadavků se zadavatelem, kde zadavatel práce spatřil několik drobných nedostatků či možných vylepšení. S nejvyšší prioritou by mělo být řešeno přidání kratšího intervalu automatické synchronizace, například na 10 minut. Dále by bylo

vhodné rozšířit možnosti správy profilu o změnu e-mailu a zrušení účtu uživatelem. V souvislosti s konfigurací spojení pak by mělo dojít k rozšíření o nastavení notifikací u každého spojení zvlášť. V neposlední řadě by někteří uživatelé jistě uvítali uživatelský manuál.

6.2 Krátkodobý výhled

Dalším cílem bude dokončení aplikace dle požadavků vydefinovaných v analytické části této práce. Tyto požadavky lze přesto rozdělit dle jejich priority do dvou kategorií. Tato sekce se zabývá těmi s vyšší prioritou, které by bylo vhodné realizovat v krátkodobém časovém horizontu.

Za nejvýznamnější budoucí rozšíření lze považovat přidání nástroje Jira mezi podporované nástroje k synchronizaci a současný návrh synchronizace časových záznamů mezi dvěma projektovými nástroji. Tento cíl byl jedním z požadavků zadavatele a byl důkladně analyzován již v této diplomové práci, kde v podkapitole Integrace nástroje Jira lze najít informace o očekávaném chování při synchronizaci. Výstupem této části analýzy jsou pak požadavky P23–P25. Vzhledem k vyšší náročnosti a celkovému rozsahu tohoto cíle může být vhodné toto rozšíření nástroje Timer2Ticket taktéž zadat jakožto téma budoucí závěrečné práce. V této souvislosti se předpokládá, že participace na projektu bude probíhat i po dokončení této diplomové práce. Částečně tomu je již nyní, kdy dochází ke spolupráci s Bc. Janem Starůstkou, který v rámci své diplomové práce bude pokračovat na vývoji nástroje Timer2Ticket.

Dalším možném vylepšením aplikace se týká pluginu Vuex, který slouží jako centrální úložiště pro všechny komponenty ve frontendové aplikaci. V současné verzi frontend při vstupu do aplikace provede načtení dat z databáze, která jsou následně uložena v pluginu Vuex a poté zobrazena uživateli. Bohužel se to však děje při každém vstupu do aplikace uživatelem, jelikož jsou uložená data po opuštění nebo obnovení stránky obnovena. [106]

Pro lepší uživatelský komfort by bylo vhodné data ukládat například do lokálního úložiště webového prohlížeče nebo do cookies tak, aby nebylo nutné data načítat před každým vstupem na stránku. Využitím lokálního úložiště by bylo možné provádět načítání aktuálních dat na pozadí, čímž by se zrychlilo zobrazení webové stránky uživateli. Trvalým uložením dat z pluginu Vuex do lokálního úložiště prohlížeče se zabývá například článek [107] nebo existuje plugin umožňující pracovat s daty jako s pluginem Vuex, který ukládá data trvale [108].

Z důvodu bezpečnosti aplikace, respektive potenciálních zranitelností, by taktéž bylo vhodné povýšit veškeré použité knihovny a balíčky na aktuální verze, či alespoň na verze bez zranitelností. To však je již vydefinováno v seznamu požadavků pod číslem P27.

Poslední oblastí, kterou se bude tato podkapitola zabývat, je podpora filtrování, stránkování a řazení při dotazování backendu. Na stránkách spojení a logy lze zobrazit teoreticky nekonečně mnoho objektů. Podpora filtrování a stránkování by pak zvýšila uživatelský komfort a tato optimalizace by zvýšila efektivitu a rychlost aplikace.

Co se týče spojení, zde nelze očekávat velké množství nakonfigurovaných spojení, současná verze ani nepodporuje filtrování spojení. Z tohoto důvodu prozatím u spojení není nutné přidávat podporu filtrování ani stránkování, i když v budoucnu je možné, že bude nutné filtrování spojení uživatele přidat. Bc. Jakub Čermák ve své bakalářské práci [15] navrhuje možnost vlastního pojmenování boxů, dle kterého by bylo možné je filtrovat. Jeden jím z vytvořených úkolů v nástroji Redmine navíc mluví o možnosti přidání řazení boxů spojení dle potřeb uživatele. V případě řazení pak bude nutné zvážit, zdali je dostatečné řazení provádět až ve frontendové části nástroje, či již při dotazování databáze backendem.

V případě logů je situace jiná. Nyní je filtrování, stránkování a řazení prováděno na straně frontendu, po získání všech příslušných dat k zobrazení. V případě logů jobů sice dochází k jejich mazání, jsou-li starší 90 dnů, avšak při případné synchronizaci objektů každou hodinu se uživatel dostane k několika tisícům záznamů za tuto dobu, což je poměrně velké množství přenesených dat. Z tohoto důvodu by bylo vhodné, aby koncový bod vracející logy jobů uživatele umožňoval filtrování, stránkování a řazení dat.

Podobná situace je v případě logů použitých okamžitých synchronizací. Zde se navíc záznamy v databázi nemažou, jejich počet tak může velmi rychle narůstat. I tento koncový bod by tedy měl podporovat filtrování, stránkování a řazení dat.

6.3 Dlouhodobý výhled

Do dlouhodobého výhledu pak patří ostatní požadavky s nižší prioritou, které nebyly realizovány, kdy jejich převážná většina pochází z bakalářské práce Bc. Jakuba Čermáka [15]. Jedná se o dokončení nového frontendu, což zahrnuje například úpravu grafických prvků či přidání funkcí poskytující vyšší uživatelský komfort při používání aplikace. Cílem zůstává i dokončení propagačního webu. Pro obojí výše zmíněné existují v nástroji Redmine již dříve vytvořené úkoly.

Jedním z cílů by v budoucnu mělo být též přidání podpory webhooků, pokud je podporované nástroje umožňují využívat. Toto téma bylo stejně tak analyzováno v této práci a náleží mu požadavek P30. S přidáním podpory webhooků by se zvýšila efektivita, jelikož by bylo možné reagovat přímo na prováděné změny v nástroji, čímž by odpadla nutnost při každé synchronizaci načíst všechny synchronizované objekty ze služeb k synchronizaci a jejich zpracování. Navíc by použití webhooků bylo možné zohlednit v nabídce členství, která by mohla nabízet například okamžitou synchronizaci.

Dalším cílem, kterému bude v budoucnu nutné se věnovat, je navrhnutí synchronizace mezi nástroji pro měření času. Původní verze realizovala pouze synchronizaci mezi Redmine a Toggl Track, tedy mezi projektovým nástrojem a nástrojem pro měření času. Tato diplomová práce analyzovala možnou synchronizaci mezi dvěma projektovými nástroji, logicky tak zbývá návrh synchronizace mezi nástroji pro měření času.

Synchronizace mezi nástroji pro měření času je důležitá především pro univerzálnost nástroje Timer2Ticket, který by měl umožňovat synchronizaci mezi všemi podporovanými nástroji. Bude-li tedy do Timer2Ticket přidán nějaký další nástroj pro měření času, bude nutné nejprve vyřešit toto téma, teprve poté bude možné přidat podporu samotného nástroje.

Poslední dlouhodobý cíl je spatřen v možnosti rozšiřování nabídky nástrojů, mezi kterými lze provádět synchronizaci. Několik jednotek projektových nástrojů a nástrojů pro měření času je popsáno v závěrečných pracích věnujících se nástroji Timer2Ticket [12, 15], avšak obecně lze říci, že v tomto směru jsou možnosti rozšiřování prakticky neomezené.

Závěr

Tato práce se zabývala dokončením již existujícího synchronizačního nástroje Timer2Ticket tak, aby bylo možné její produkční nasazení. Tohoto primárního cíle bylo dosaženo tak, že byl nejprve popsán stav současné verze nástroje, která byla implementována v rámci diplomové práce Ing. Víta Štefana a bakalářské práce Bc. Jakuba Čermáka. Následně byly analyzovány požadavky na rozšíření nástroje. Prvním zdrojem požadavků byly výše zmíněné práce, ve kterých autoři sami navrhli možné směřování nástroje do budoucna. Druhým zdrojem pak byl zadavatel této diplomové práce, který má taktéž své zájmy v budoucím využívání nástroje.

Na základě nabytých poznatků vzešly konkrétní požadavky na nástroj Timer2Ticket, kterým však z důvodu jejich velkého počtu byla stanovena priorita. Pro větší přehlednost byly související požadavky taktéž seskupeny do větších logických celků. Nejvyšší prioritou bylo ohodnoceno napojení nového frontendu na backend, zprovoznění monetizace, oddělení komerční a nekomerční části, povýšení Toggl Track API a zpětná kompatibilita s původní verzí.

Následně byly navrženy potřebné změny a rozšíření, které byly provedeny pro požadavky s nejvyšší prioritou. Současně byly upraveny již existující případy užití a rozšířen původní návrh architektury nástroje. Dle vzniklého návrhu pak došlo k implementaci. Implementace probíhala ve všech třech stávajících částech nástroje, se snahou zachovat použité nástroje, knihovny a technologie. Navíc došlo k přidání dvou knihoven pro část T2T API a pro nový frontend. Další komponentou je pak vytvoření migračního skriptu tak, aby bylo možné data z původní verze aplikace používat v nové verzi. Implementovaná aplikace byla následně nasazená a je veřejně dostupná na doméně `https://new.timer2ticket.com`.

Implementace rozšíření nástroje Timer2Ticket pak byla podrobena několika typům testů. Otestován byl zdrojový kód pomocí jednotkových testů. Dále byly otestovány koncové body nástroje a následně celá aplikace s reálnými daty. Poté proběhlo akceptační testování se zadavatelem práce, který i přes drobné výhrady rozšířený nástroj přijal. Dále, již mimo rámec této diplomové práce, bude pokračovat interní testování po nasazení ve společnosti Jagu s.r.o.

Veškeré body zadání byly splněny. Vzhledem k velkému rozsahu požadavků na aplikaci a často vysoké složitosti však nebylo realizováno všech 51 vydefinovaných požadavků. Při realizaci bylo postupováno dle stanovené priority. Takto bylo postupováno se souhlasem zadavatele, se kterým byly priority jednotlivých požadavků diskutovány a následně schváleny. Implementováno bylo 26 požadavků. Zbývající požadavky nemají vysokou prioritu, nejsou tedy klíčové pro bezproblémový chod nástroje Timer2Ticket nebo se jedná o tzv. „nice to have“ požadavky. V rámci této diplomové práce však byly některé požadavky analyzovány či došlo k částečné implementaci. K realizaci však již právě z důvodu buďto nízké priority nebo vysoké náročnosti nedošlo.

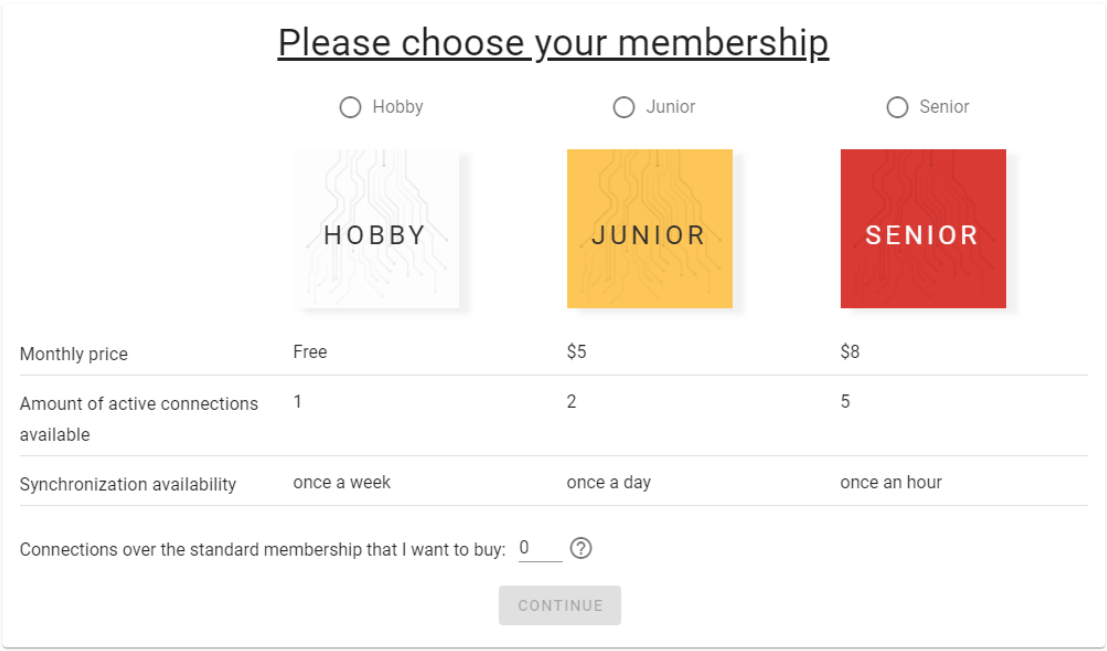
Výstupem práce je tedy verze nástroje Timer2Ticket s plně funkčním novým frontendem tak, že je možné její produkční nasazení. To mimo jiné dokazuje fakt, že je aplikace nasazená. Implementace mimo to integruje jednotnou platformu pro přihlašování a registraci uživatelů, podporuje více spojení pro jednoho uživatele nebo volbu časové zóny pro plán synchronizace.

Také došlo ke zprovoznění modelu monetizace zahrnující volbu členství uživatele včetně přikoupení spojení nad rámec členství a jeho správu nebo podporu okamžitých synchronizací. Pro možnost platby byl nástroj napojen na platební bránu a uživateli je umožněno zobrazit si historii objednávek či uložení platebních a fakturačních údajů. Verzi je možné používat jak v komerční, tak i nekomerční verzi a je umožněno převést data ze staré verze.

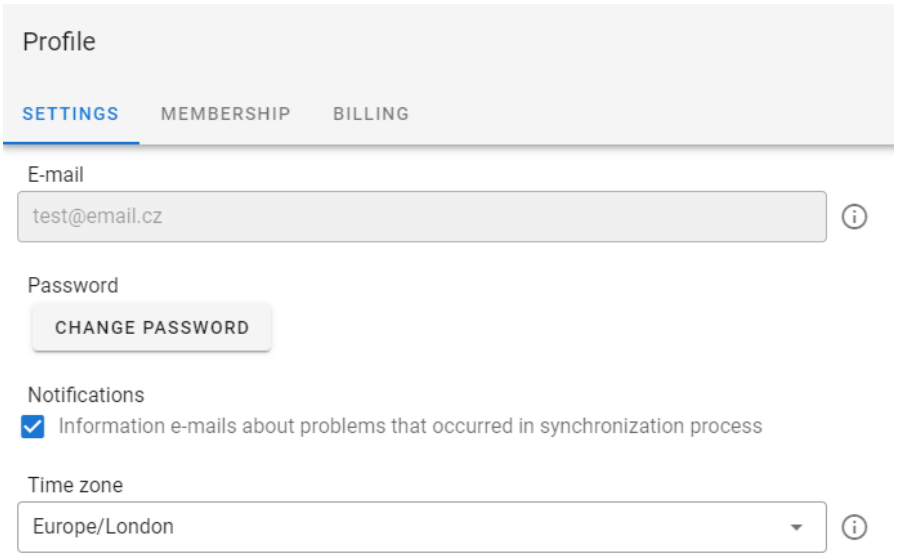
V závěru je popsáno možné budoucí směřování a rozvoj nástroje Timer2Ticket. Bezprostředně navazující rozšíření popisují budoucí vývoj reagující na tuto diplomovou práci. V rámci krátkodobých cílů je doporučen rozvoj především v rozšíření podpory integrovaných nástrojů, zejména pak Jiry, a návrhu synchronizace časových záznamů mezi dvěma nástroji pro měření času. Dále by bylo vhodné provést úpravy frontendu navržené již Bc. Jakubem Čermákem. Dlouhodobé cíle jsou pak spatřeny v dokončení propagačního webu a zvýšení efektivity při synchronizaci.

Příloha A

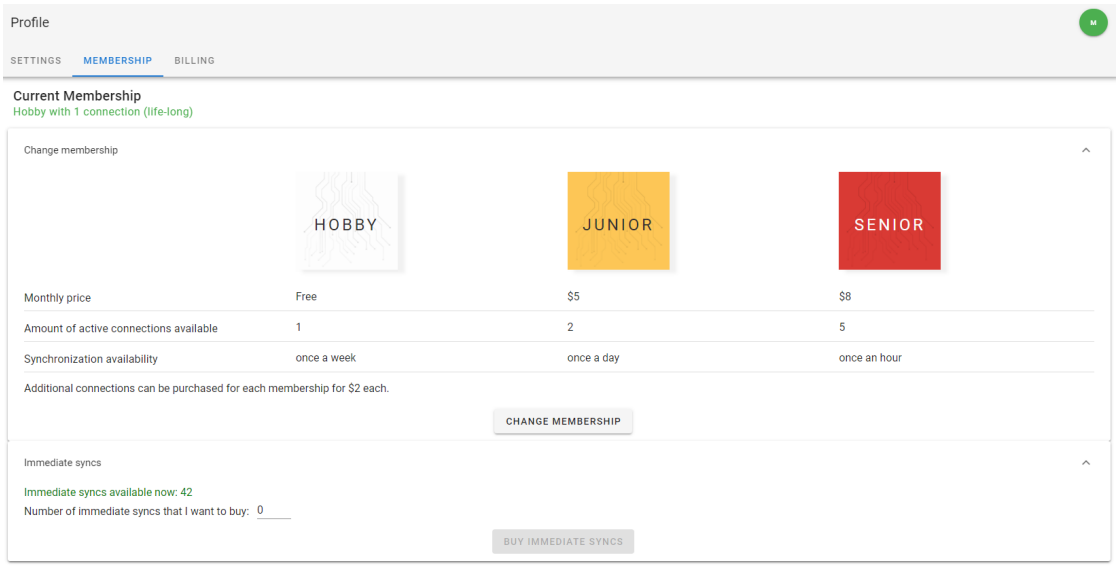
Ukázka provedených změn



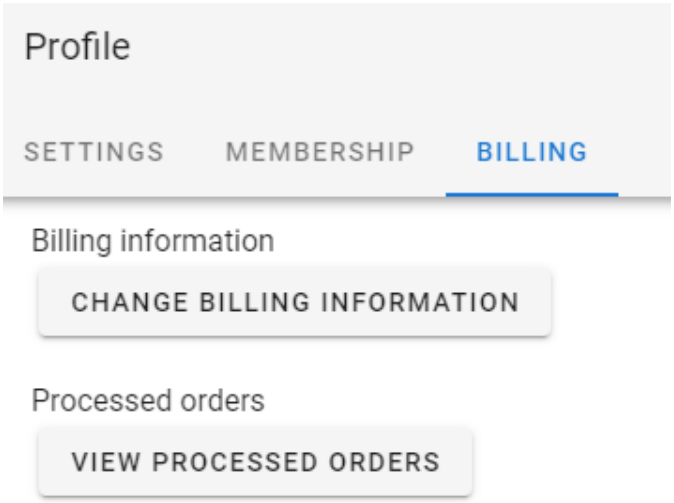
Obrázek A.1 Ukázka tabulky s volbou členství, není-li zatím uživatelem zvolené



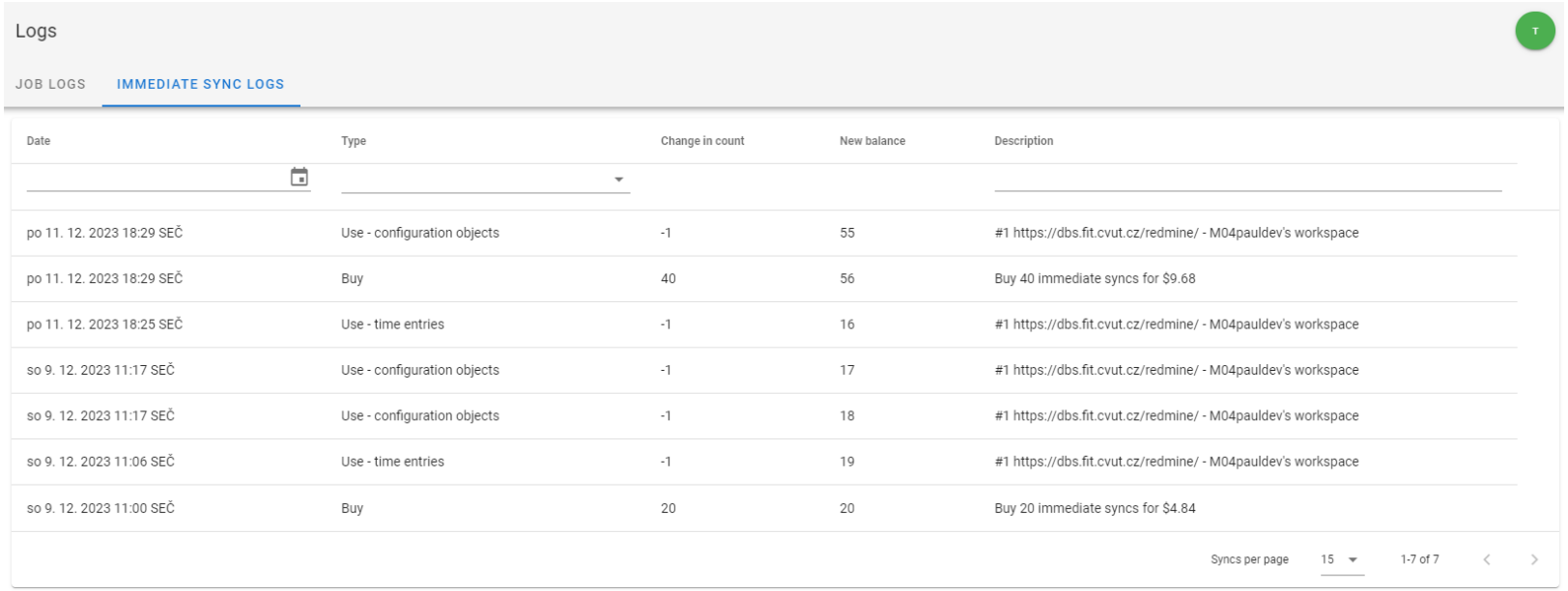
Obrázek A.2 Ukázka obrazovky podsekce Nastavení sekce Profil



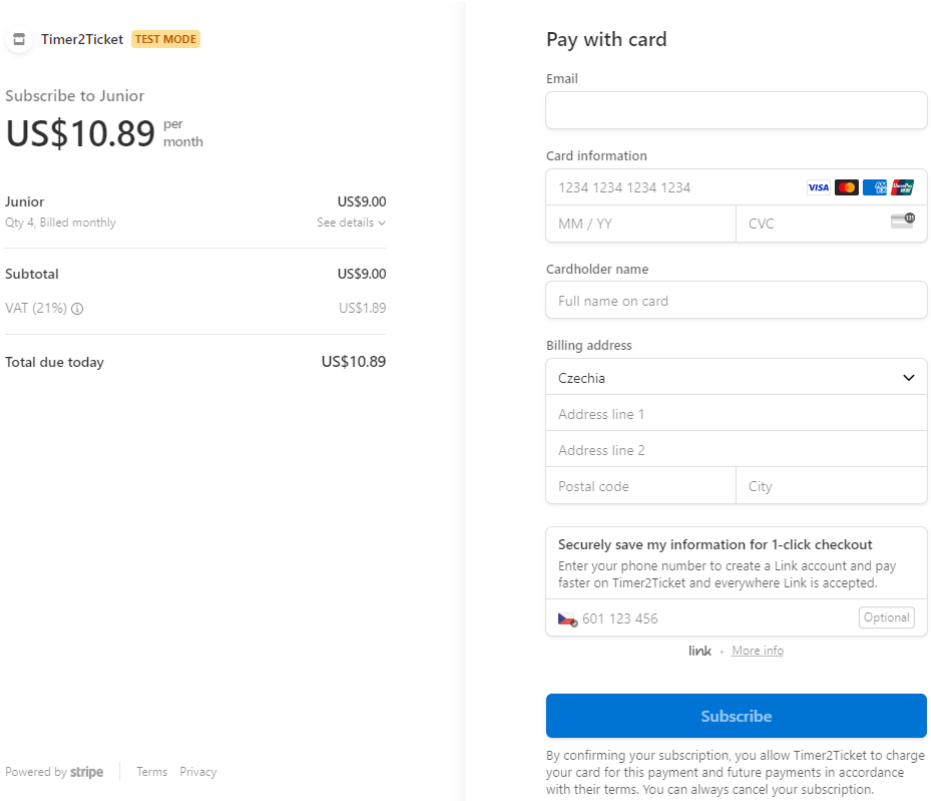
Obrázek A.3 Ukázka obrazovky podsekce Členství sekce Profil



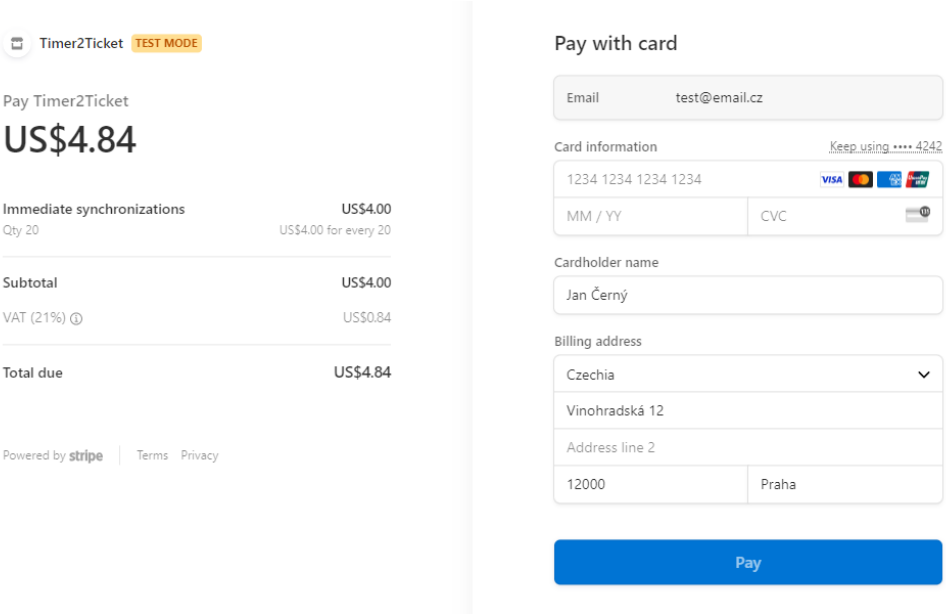
Obrázek A.4 Ukázka obrazovky podsekce Platby sekce Profil



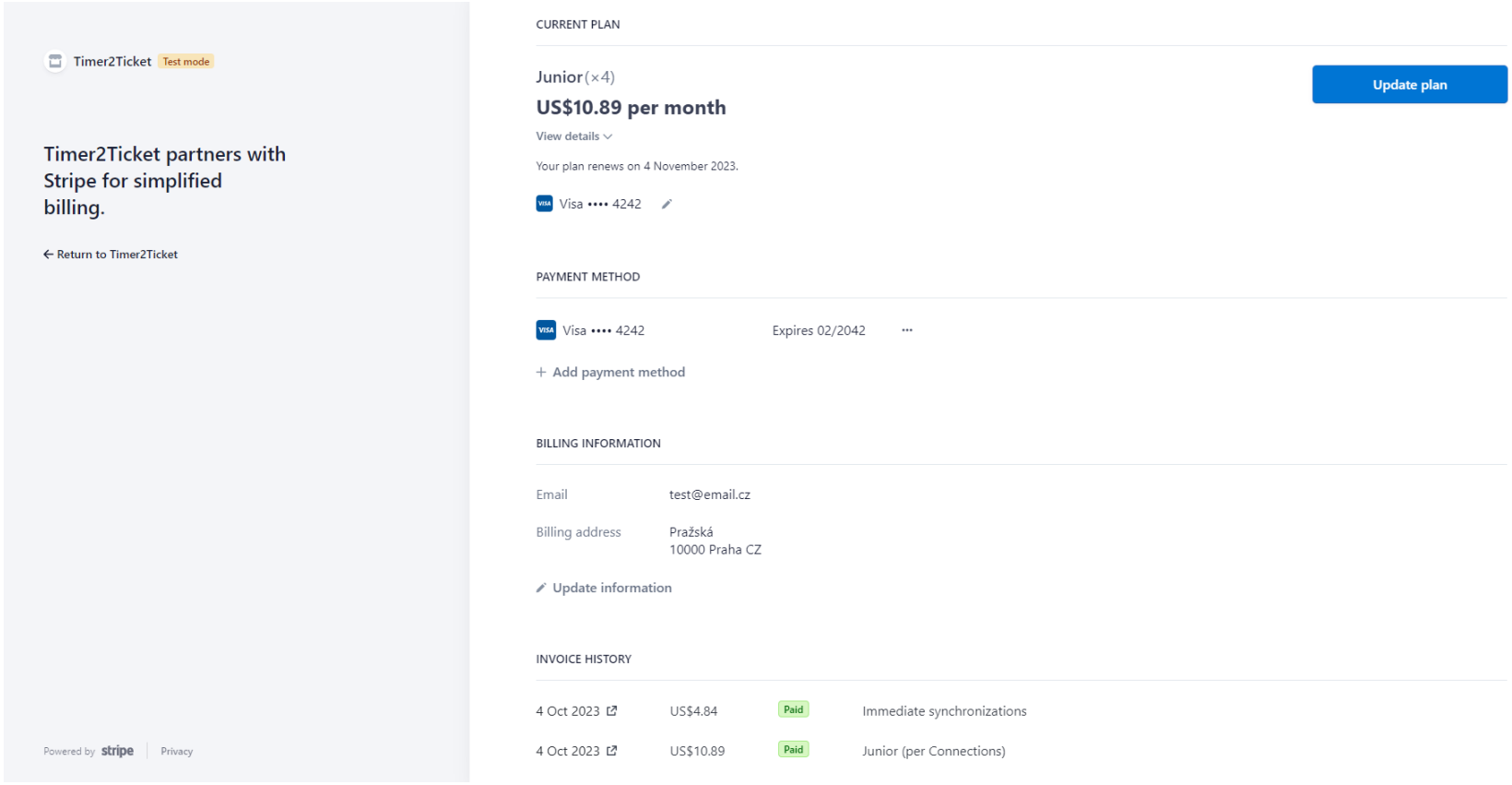
Obrázek A.5 Ukázka obrazovky podsekce Logy okamžitých synchronizací sekce Logy



Obrázek A.6 Ukázka obrazovky platební brány pro uhrazení zvoleného členství



Obrázek A.7 Ukázka obrazovky platební brány pro nákup okamžitých synchronizací



Obrázek A.8 Ukázka obrazovky zákaznického portálu platební brány Stripe

Literatura

1. ZOUBEK, Bohumír. *BI-SI2.3 – Softwarové inženýrství 2*. Odhady, nabídky, měření a historie [online]. 2021. [cit. 2023-02-20]. Dostupné z: https://moodle-vyuka.cvut.cz/pluginfile.php/530559/course/section/84272/2021_2022/02_ProposalsEstimationsMeasurement.pdf.
2. *Toggl*. Time Tracking Software, Project Planning & Hiring tools [online]. 2023. [cit. 2023-02-20]. Dostupné z: https://toggl.com/.
3. *Everhour*. Everhour – Free Time Tracker & Timesheets for Teams [online]. 2023. [cit. 2023-02-20]. Dostupné z: https://everhour.com/.
4. *Timely*. Timely - Time Tracking Software [online]. 2023. [cit. 2023-02-20]. Dostupné z: https://timelyapp.com/hp-tc.
5. *TopTracker*. TopTracker - Free Time Tracking, Invoicing & Payments App [online]. 2022. [cit. 2023-02-20]. Dostupné z: https://www.toptal.com/tracker.
6. *Atlassian*. Jira Work Management | A Friendly and Powerful Way to Work [online]. 2022. [cit. 2023-02-20]. Dostupné z: https://www.atlassian.com/software/jira/work-management.
7. *Atlassian*. Jira | Issue & Project Tracking Software | Atlassian [online]. 2023. [cit. 2023-02-20]. Dostupné z: https://www.atlassian.com/software/jira.
8. LANG, Jean-Philippe. *Redmine*. Overview - Redmine [online]. 2014. [cit. 2023-02-20]. Dostupné z: https://www.redmine.org/.
9. *JetBrains*. YouTrack: Project Management for all your teams [online]. 2023. [cit. 2023-02-20]. Dostupné z: https://www.jetbrains.com/youtrack/.
10. *monday.com*. monday.com | A new way of working [online]. 2023. [cit. 2023-02-20]. Dostupné z: https://monday.com/.
11. *ClickUp*. ClickUp™ | One app to replace them all [online]. 2023. [cit. 2023-02-20]. Dostupné z: https://clickup.com/.
12. 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.
13. *GitHub*. Timer2Ticker [online]. 2023. [cit. 2023-03-02]. Dostupné z: https://github.com/orgs/Timer2Ticket/repositories.
14. *GitHub*. timer2ticket-core/LICENSE.md [online]. 2021. [cit. 2023-06-18]. Dostupné z: https://github.com/Timer2Ticket/timer2ticket-core/blob/main/LICENSE.md.

15. ČERMÁK, Jakub. *Synchronizační middleware mezi projektovým systémem a aplikací na sledování času stráveného na projektech*. 2022. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
16. *Microsoft*. Typescript: JavaScript with syntax for types. [online]. 2023. [cit. 2023-02-27]. Dostupné z: https://www.typescriptlang.org/.
17. *OpenJS Foundation*. Node.js [online]. 2023. [cit. 2023-02-27]. Dostupné z: https://www.typescriptlang.org/.
18. *OpenJS Foundation*. Express - Node.js web application framework [online]. 2017. [cit. 2023-02-27]. Dostupné z: https://expressjs.com/.
19. *GitHub*. ladjs/superagent: Ajax for Node.js and browsers (JS HTTP client). Maintained for @forwardemail, @ladjs, @spamscanner, @breejs, @cabinjs, and @lassjs. [online]. 2023. [cit. 2023-02-27]. Dostupné z: https://github.com/ladjs/superagent.
20. *GitHub*. typestack/class-validator: Decorator-based property validation for classes. [online]. 2023. [cit. 2023-02-27]. Dostupné z: https://github.com/typestack/class-validator.
21. *GitHub*. kelektiv/node-cron: Cron for NodeJS. [online]. 2023. [cit. 2023-02-27]. Dostupné z: https://github.com/kelektiv/node-cron.
22. LANG, Jean-Philippe. *Redmine*. Rest api - Redmine [online]. 2014. [cit. 2023-02-20]. Dostupné z: https://www.redmine.org/projects/redmine/wiki/rest_api.
23. *Toggl*. Introduction to workspaces | Toggl Track Knowledge Base [online]. 2023. [cit. 2023-02-22]. Dostupné z: https://support.toggl.com/en/articles/2452474-introduction-to-workspaces.
24. *Toggl*. Toggl Track Categorization Guide | Toggl Track Knowledge Base [online]. 2023. [cit. 2023-02-22]. Dostupné z: https://support.toggl.com/en/articles/4200664-toggl-track-categorization-guide.
25. *Toggl*. Tags | Toggl Track Knowledge Base [online]. 2023. [cit. 2023-02-22]. Dostupné z: https://support.toggl.com/en/articles/2220689-tags.
26. *Toggl*. Toggl Track API | Track Your Way [online]. 2023. [cit. 2023-02-20]. Dostupné z: https://developers.track.toggl.com/.
27. AGUIAR, Lucas. *Toggl*. Announcing the new Toggl Track Reports API, v3 [online]. 2022. [cit. 2023-02-27]. Dostupné z: https://toggl.com/blog/toggl-track-reports-api-v3.
28. *Toggl*. Blog | Track Your Way [online]. 2023. [cit. 2023-02-22]. Dostupné z: https://developers.track.toggl.com/changes/.
29. *GitHub*. toggl/toggl_api_docs: Documentation for the Toggl API [online]. 2022. [cit. 2023-02-25]. Dostupné z: https://github.com/toggl/toggl_api_docs.
30. *Oracle*. Cron Expressions [online]. 2009. [cit. 2023-03-02]. Dostupné z: https://docs.oracle.com/cd/E12058_01/doc/doc.1014/e12030/cron_expressions.htm.
31. *Google*. Angular [online]. 2023. [cit. 2023-02-27]. Dostupné z: https://angular.io/.
32. *RxJS*. RxJS - Observable [online]. 2023. [cit. 2023-02-27]. Dostupné z: https://rxjs.dev/guide/observable.
33. *Jagu s.r.o.* Timer2Ticker Client [online]. 2021. [cit. 2023-03-02]. Dostupné z: https://app.timer2ticket.com/.
34. YOU, Evan. *Vue.js*. Vue.js - The Progressive JavaScript Framework | Vue.js [online]. 2023. [cit. 2023-03-04]. Dostupné z: https://vuejs.org/.

35. YOU, Evan. *Vue.js*. Overview | Vue CLI [online]. 2018. [cit. 2023-03-04]. Dostupné z: https://cli.vuejs.org/guide/.
36. *Sentry*. Application Performance Monitoring & Error Tracking Software | Sentry [online]. 2023. [cit. 2023-03-04]. Dostupné z: https://sentry.io/welcome/.
37. BOBKOV, Alexey; GEORGES, Samuel. *October CMS*. October - PHP CMS platform based on the Laravel Framework [online]. 2023. [cit. 2023-03-04]. Dostupné z: https://octobercms.com/.
38. *Jagu s.r.o.* Timer2Ticket - Demonstration [online]. 2022. [cit. 2023-03-04]. Dostupné z: https://timer2ticket.com/.
39. *Okta, Inc.* Auth0: Secure access for everyone. But not just anyone. [online]. 2023. [cit. 2023-03-08]. Dostupné z: https://auth0.com/.
40. LANG, Jean-Philippe. *Redmine*. Developer guide [online]. 2014. [cit. 2023-03-18]. Dostupné z: https://www.redmine.org/projects/redmine/wiki/Developer_Guide.
41. LANG, Jean-Philippe. *Redmine*. Redmine plugin hooks [online]. 2014. [cit. 2023-03-18]. Dostupné z: https://www.redmine.org/projects/redmine/wiki/Hooks.
42. *GitHub*. Redmine WebHook Plugin [online]. 2022. [cit. 2023-03-18]. Dostupné z: https://github.com/suer/redmine_webhook.
43. *GitHub*. Redmine Plugin Webhook [online]. 2021. [cit. 2023-03-18]. Dostupné z: https://github.com/ostrovok-team/redmine-plugin-webhook.
44. *Atlassian*. Webhooks [online]. 2023. [cit. 2023-03-18]. Dostupné z: https://developer.atlassian.com/server/jira/platform/webhooks/.
45. *GitHub*. JsonWebToken implementation for node.js [online]. 2023. [cit. 2023-04-08]. Dostupné z: https://github.com/auth0/node-jsonwebtoken.
46. *Red Hat*. Keycloak [online]. 2023. [cit. 2023-04-10]. Dostupné z: https://www.keycloak.org/.
47. WAGDE, Sampada. *Baeldung*. Customizing the Login Page for Keycloak [online]. 2023. [cit. 2023-04-10]. Dostupné z: https://www.baeldung.com/keycloak-custom-login-page.
48. *Stripe, Inc.* Stripe - Payment Processing Platform for the Internet [online]. 2023. [cit. 2023-06-06]. Dostupné z: https://stripe.com.
49. *Stripe, Inc.* Stripe API reference [online]. 2023. [cit. 2023-06-05]. Dostupné z: https://stripe.com/docs/api.
50. *Stripe, Inc.* Stripe Documentation [online]. 2023. [cit. 2023-06-04]. Dostupné z: https://stripe.com/docs.
51. *Stripe, Inc.* Accelerate growth with Stripe Billing [online]. 2023. [cit. 2023-06-09]. Dostupné z: https://stripe.com/en-cz/billing.
52. CHU, Theodora. *Stripe, Inc.* Introducing the Billing customer portal [online]. 2020. [cit. 2023-06-09]. Dostupné z: https://stripe.com/blog/billing-customer-portal.
53. *Stripe, Inc.* Stripe Tax: Automate tax collection on your Stripe transactions [online]. 2023. [cit. 2023-06-07]. Dostupné z: https://stripe.com/en-cz/tax.
54. *Stripe, Inc.* Tax calculations [online]. 2023. [cit. 2023-06-07]. Dostupné z: https://stripe.com/docs/tax/calculating.
55. *Fakturoid s.r.o.* Faktury do zahraničí – náležitosti, zvyklosti a doporučení [online]. 2023. [cit. 2023-06-07]. Dostupné z: https://www.fakturoid.cz/almanach/zacatky-podnikani/faktury-do-zahranici.

56. *Fakturoid s.r.o.* Osvobození od DPH [online]. 2023. [cit. 2023-06-07]. Dostupné z: https://www.fakturoid.cz/almanach/dane/osvobozeni-od-dph.
57. *Atlassian.* Jira Software vs Redmine Comparison | Atlassian [online]. 2022. [cit. 2023-03-14]. Dostupné z: https://www.atlassian.com/software/jira/comparison/jira-vs-redmine.
58. MACKAY, Jory. *Planio GmbH.* Redmine versus Jira: How to pick the right tool for your team [online]. 2022. [cit. 2023-03-14]. Dostupné z: https://plan.io/blog/redmine-versus-jira/.
59. *Datanyze.* Project Management Software Market Share [online]. 2023. [cit. 2023-03-14]. Dostupné z: https://www.datanyze.com/market-share/project-management--217.
60. *Atlassian.* The Jira Cloud platform REST API [online]. 2023. [cit. 2023-03-16]. Dostupné z: https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/.
61. *Atlassian.* Manage API tokens for your Atlassian account [online]. 2023. [cit. 2023-03-17]. Dostupné z: https://support.atlassian.com/atlassian-account/docs/manage-api-tokens-for-your-atlassian-account/.
62. *Atlassian.* Use advanced search with Jira Query Language (JQL) [online]. 2023. [cit. 2023-03-18]. Dostupné z: https://support.atlassian.com/jira-service-management-cloud/docs/use-advanced-search-with-jira-query-language-jql/.
63. *Fakturoid s.r.o.* Jednodušší a chytřejší podnikání pro vás [online]. 2023. [cit. 2023-06-09]. Dostupné z: https://www.fakturoid.cz/.
64. *Fakturoid s.r.o.* Fakturoid pro vývojáře [online]. 2023. [cit. 2023-06-09]. Dostupné z: https://www.fakturoid.cz/api.
65. KONAROVSKÝ, Lukáš. *Oracle.* Fakturoid API v2 [online]. 2023. [cit. 2023-06-09]. Dostupné z: https://fakturoid.docs.apiary.io/#.
66. *ProductPlan.* MoSCoW Prioritization [online]. 2023. [cit. 2023-07-12]. Dostupné z: https://www.productplan.com/glossary/moscow-prioritization/.
67. GEKHT, Nikolay. *Gehtsoft USA.* Create Better Backlog and Engage the Development Team with FURPS. [online]. 2020. [cit. 2023-07-12]. Dostupné z: https://gehtsoftusa.com/blog/create-better-backlog-and-engage-the-development-team-with-furps/.
68. ZIEMEK, Marcin. *Marcin Ziemek.* Documenting non-functional requirements using FURPS+ [online]. 2022. [cit. 2023-09-03]. Dostupné z: https://www.marcinziemek.com/blog/content/articles/8/article_en.html.
69. OTTINGER, Tim; LANGR, Jeff. *Agile in a Flash.* FURPS+ [online]. 2020. [cit. 2023-07-12]. Dostupné z: http://agileinaflash.blogspot.com/2009/04/furps.html.
70. *Sparx Systems Pty Ltd.* Enterprise Architect [online]. 2023. [cit. 2023-06-04]. Dostupné z: https://sparxsystems.com/products/ea/index.html.
71. MCGRATH, Adam. *Okta, Inc.* Introducing the OAuth 2.0 Express SDK for Protecting APIs with JWT Bearer Tokens [online]. 2022. [cit. 2023-06-05]. Dostupné z: https://auth0.com/docs/libraries/auth0-single-page-app-sdk.
72. *Stripe, Inc.* Set up Stripe Tax [online]. 2023. [cit. 2023-06-05]. Dostupné z: https://stripe.com/docs/tax/set-up.
73. *MongoDB, Inc.* JSON and BSON [online]. 2023. [cit. 2023-10-10]. Dostupné z: https://www.mongodb.com/json-and-bson.
74. *JGraph Ltd.* Flowchart Maker & Online Diagram Software [online]. 2023. [cit. 2023-06-03]. Dostupné z: https://app.diagrams.net/.

75. *Toggl*. v8 Migration Guide [online]. 2023. [cit. 2023-03-27]. Dostupné z: https://developers.track.toggl.com/docs/additional/v8_migration_guide/index.html.
76. ARIAS, Dan. *Okta, Inc.* The Complete Guide to Vue.js User Authentication with Auth0 [online]. 2022. [cit. 2023-06-04]. Dostupné z: https://auth0.com/blog/complete-guide-to-vue-user-authentication/.
77. *Okta, Inc.* Add authorization to an Express.js API application [online]. 2023. [cit. 2023-06-04]. Dostupné z: https://auth0.com/docs/quickstart/backend/nodejs/interactive.
78. *Okta, Inc.* Authentication API Explorer [online]. 2023. [cit. 2023-06-06]. Dostupné z: https://auth0.com/docs/api/authentication#user-profile.
79. *Okta, Inc.* Auth0 Management API v2 [online]. 2023. [cit. 2023-06-06]. Dostupné z: https://auth0.com/docs/api/management/v2.
80. GORE, Anthony. *LogRocket, Inc.* Building a Vue 3 component library [online]. 2021. [cit. 2023-06-10]. Dostupné z: https://blog.logrocket.com/building-vue-3-component-library/.
81. *npm, Inc.* npm-link [online]. 2021. [cit. 2023-06-10]. Dostupné z: https://docs.npmjs.com/cli/v9/commands/npm-link.
82. *npm, Inc.* Creating and publishing private packages [online]. 2021. [cit. 2023-06-10]. Dostupné z: https://docs.npmjs.com/creating-and-publishing-private-packages.
83. *Mozilla Foundation*. import() - JavaScript [online]. 2023. [cit. 2023-06-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/import.
84. BEHRENS, Mattie. *Atomic Object LLC*. Build an NPM Package in TypeScript from the Ground Up [online]. 2022. [cit. 2023-06-10]. Dostupné z: https://spin.atomicobject.com/2022/06/21/npm-package-typescript/.
85. *MongoDB, Inc.* Class: Collection [online]. 2021. [cit. 2023-07-06]. Dostupné z: https://mongodb.github.io/node-mongodb-native/3.6/api/Collection.html.
86. *Stack Exchange Inc*. How to get list of all timezones in javascript [online]. 2023. [cit. 2023-06-09]. Dostupné z: https://stackoverflow.com/questions/38399465/how-to-get-list-of-all-timezones-in-javascript.
87. *GitHub*. node-cron/node-cron: A simple cron-like job scheduler for Node.js [online]. 2021. [cit. 2023-06-26]. Dostupné z: https://github.com/node-cron/node-cron.
88. *GitHub*. motdotla/dotenv: Loads environment variables from .env for nodejs projects. [online]. 2021. [cit. 2023-06-26]. Dostupné z: https://github.com/motdotla/dotenv.
89. *GitHub, Inc.* Working with the npm registry [online]. 2023. [cit. 2023-09-30]. Dostupné z: https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-npm-registry.
90. *Docker Inc.* Docker Compose overview [online]. 2023. [cit. 2023-10-29]. Dostupné z: https://docs.docker.com/compose/.
91. *OpenJS Foundation*. Mocha - the fun, siimple, flexible JavaScript test framework [online]. 2023. [cit. 2023-07-05]. Dostupné z: https://mochajs.org/.
92. *Chai Assertion Library*. Chai Assertion Library [online]. 2023. [cit. 2023-07-05]. Dostupné z: https://www.chaijs.com/.
93. *Chai Assertion Library*. Sinon.JS - Standalone test spies, stubs and mocks for JavaScript. Works with any unit testing framework. [online]. 2023. [cit. 2023-07-05]. Dostupné z: https://www.chaijs.com/.

94. SCHWERING, Ramona; YEEN, Jecelyn. *web.dev*. Four common types of code coverage [online]. 2023. [cit. 2023-10-01]. Dostupné z: https://web.dev/ta-code-coverage/.
95. *Istanbul*. JavaScript test coverage made simple. [online]. 2023. [cit. 2023-10-01]. Dostupné z: https://istanbul.js.org/.
96. *GitHub*. nyc [online]. 2021. [cit. 2023-10-01]. Dostupné z: https://github.com/istanbuljs/nyc.
97. *Istanbul*. Using Istanbul With TypeScript & mocha [online]. 2023. [cit. 2023-10-01]. Dostupné z: https://istanbul.js.org/docs/tutorials/typescript/.
98. LUCERO, Carlos. *Medium.com*. Test coverage report in Typescript with Mocha and NYC [online]. 2021. [cit. 2023-10-01]. Dostupné z: https://medium.com/@ocnvn/test-coverage-report-in-typescript-with-mocha-and-nyc-a6b10cbec24.
99. *GitHub*. c8 - native V8 code-coverage [online]. 2023. [cit. 2023-10-01]. Dostupné z: https://github.com/bcoe/c8.
100. *GitHub*. Proposal: Use Node's native coverage when available or specified #1343 [online]. 2021. [cit. 2023-10-01]. Dostupné z: https://github.com/istanbuljs/nyc/issues/1343.
101. PITTET, Sten. *Atlassian*. What is code coverage? [online]. 2023. [cit. 2023-10-04]. Dostupné z: https://www.atlassian.com/continuous-delivery/software-testing/code-coverage.
102. KOVÁŘ, Pavel. *Automatizované testování webového portálu dbs.fit.cvut.cz*. 2017. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
103. *BI-SI2.3 – Softwarové inženýrství 2*. Testování [online]. 2021. [cit. 2023-02-20]. Dostupné z: https://moodle-vyuka.cvut.cz/pluginfile.php/641049/course/section/101354/2021_2022/06_Testing.pdf.
104. *Postman, Inc.* Postman API Platform [online]. 2023. [cit. 2023-07-05]. Dostupné z: https://www.postman.com/.
105. *Postman, Inc.* Using variables | Postman Learning Center [online]. 2023. [cit. 2023-07-05]. Dostupné z: https://learning.postman.com/docs/sending-requests/variables/.
106. *Vuex*. What is Vuex? [online]. 2023. [cit. 2023-06-26]. Dostupné z: https://vuex.vuejs.org/.
107. WITTMANN, Axel. *Medium.com*. How to permanently save data with Vuex & localStorage in your Vue app [online]. 2020. [cit. 2023-06-26]. Dostupné z: https://medium.com/js-dojo/how-to-permanently-save-data-with-vuex-localstorage-in-your-vue-app-f1d5c140db69.
108. *Github*. vuex-persist [online]. 2020. [cit. 2023-06-26]. Dostupné z: https://github.com/championswimmer/vuex-persist.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `timer2ticket-api` ..... zdrojové kódy implementace části API
    - `timer2ticket-client-vue` ..... zdrojové kódy implementace frontendu
    - `timer2ticket-core` ..... zdrojové kódy implementace části Core
    - `timer2ticket-migrate` ..... zdrojové kódy migračního skriptu
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF