FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Trackman - sledovací služba platformy Integromat
Student:	Dominik Kadera
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je návrh a realizace služby Trackman v platformě Integromat. Platforma Integromat je založena na architektuře microservices. Trackman umožní skrze interní rozhraní mikroslužbám zasílat události ke zpracování do fronty, kterou následně zpracuje a pro finální odeslání dat do analytických nástrojů třetích stran využije interní moduly platformy Integromat, které zastřešují práci s jejich REST API. Trackman umožní jednotně agregovat události produkované mikroslužbami backend i frontend části Integromatu.

Postupujte v těchto krocích:
- stručně popište platformu Integromat, zaměřte se na popis konceptů relevantních pro tuto práci
- shromážděte požadavky na službu včetně požadavků na API, které bude sloužit pro distribuci posbíraných dat do nástrojů třetích stran, proveďte analýzu
- navrhněte službu Trackman
- návrh implementujte, řádně otestujte a zdokumentujte.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 20. února 2021 v Praze.

Bakalářská práce

TRACKMAN

Dominik Kadera

Fakulta informačních technologií ČVUT v Praze
Katedra softwarového inženýrství
Vedoucí: Ing. Michal Valenta, Ph.D.
12. května 2021

Odkaz na tuto práci: Dominik Kadera. *Trackman*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

V prvé řadě bych chtěl poděkovat panu doktoru Michalu Valentovi za to, že se ujal vedení mé bakalářské práce, byl mi nápomocen ve věcech technických i formálních a příjemně se mi pod jeho vedením pracovalo. Také bych chtěl poděkovat CTO Integromatu, panu Patriku Šimkovi, za možnost použít Trackmana jako bakalářkou práci a za nespočet konzultací, inspirativních rozhovorů a rad, které mi dával v průběhu vývoje. Děkuji také všem kolegům z Integromatu za nesmírně přátelské prostředí, které ve firmě tvoří, neb i díky nim jsem si tvorbu Trackmana, stejně jako dalších projektů v Integromatu, velice užil. Závěrem děkuji celé mé rodině za jejich nehynoucí podporu v průběhu celého studia a za jejich soudržnost a náklonnost i v těžkých obdobích.

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. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na jejímž základě se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ustanovení § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 11. května 2021 .....

Abstrakt

Tato práce se zabývá návrhem a realizací služby Trackman, která je klíčovou komponentou v rámci interního monitoringu platformy Integromat a následné distribuce nashromážděných dat k další analýze.

Cílem bylo za prvé navrhnout architekturu celé aplikace, což zahrnovalo především databázový model a architekturu aplikační vrstvy v jazyce JavaScript, ovšem také návrh samotné komunikace se třetími stranami a to s důrazem na využití již existujících integrací, které platforma Integromat nabízí, a za druhé pak navržené řešení implementovat.

Databázová vrstva je založena na databázovém stroji PostgreSQL a využívá přístupu uložených procedur na straně databázového serveru. Aplikační vrstva je napsána v jazyce JavaScript a díky využití interních knihoven Integromatu nabízí vysokou abstrakci pohledu na data a široké možnosti jejich transformace.

Trackman je v době dokončování práce nasazen v produkčním prostředí platformy a průměrně zpracuje zhruba půl milionu požadavků denně.

Klíčová slova Konfigurovatelná monitorovací služba, Sběr a distribuce událostí, Integromat, NodeJS, PostgreSQL

Abstract

This thesis covers the software design and realization of the Trackman service in the integration platform, Integromat. Trackman is a key component of the internal monitoring system of Integromat, and also plays a major role in the further distribution of collected data on the platform.

The goal was to design the whole architecture of the service, mainly the database model and architecture of the application layer in JavaScript, as well as designing the communication between Trackman and third-party services while leveraging the existing integrations that are already offered by Integromat, and finally implementing the whole solution.

The database layer is built on top of the PostgreSQL database management system and uses the Stored Procedures pattern. The application layer is then coded in JavaScript and thanks to the internal libraries of Integromat, it offers a high level of data abstraction and a wide range of possibilities for data transformation.

At the time of finishing this thesis, Trackman is already running in the production environment, serving roughly half a million requests a day.

Keywords Configurable monitoring service, Collection and distribution of events, Integromat, NodeJS, PostgreSQL

Seznam zkratek

API	Application Programming Interface
CRON	Command Run ON
HTTP	Hyper-Text Transfer Protocol
JSON	JavaScript Object Notation
REST	REpresentational State Transfer
SID	Session ID
XML	eXtensible Markup Language

Seznam termínů

backend	Logická a databázová vrstva. Část aplikace obsahující vlastní funkcionalitu, zpracování a uložení dat.
endpoint	Koncový bod. V kontextu Trackmana se jedná o URL adresu sloužící pro interakci s API třetí strany.
event	Událost. Zpravidla má název a obsah. Popisuje, co se stalo.
frontend	Prezentační vrstva. Zabezpečuje komunikaci s uživatelem pomocí grafického rozhraní.
interface	Rozhraní. Popis formátu dat, která jsou očekávána, respektive odesílána.
landing page	Přistávací stránka. Jednoúčelová webová stránka sloužící například pro propagaci jedné konkrétní služby nebo tématu.
microservice	Mikroslužba. Komponenta aplikace nesoucí zodpovědnost za vymezenou část funkcionality, která ve spojení s ostatními takovými službami tvoří jeden funkční celek
pageview	Návštěva stránky. Jedná se o událost, která vzniká pokaždé, kdy uživatel ve svém prohlížeči zobrazí nějakou stránku.
session	Sezení. Reprezentuje uživatelskou relaci na stránce.

Úvod

Základní myšlenka mé bakalářské práce vznikla ve společnosti Integromat, kde jsem během studia na Fakultě informačních technologií ČVUT pracoval jako softwarový vývojář. Počet zákazníků Integromatu rostl každým dnem o několik stovek a bylo potřeba začít nějakým způsobem vyhodnocovat, jak se zákazníci uvnitř platformy chovají, co dělají a jaké stránky procházejí a na základě těchto událostí začít přizpůsobovat chování platformy tak, abychom jim mohli nabídnout co nejlepší uživatelský zážitek.

V době, kdy je internet plný cílené reklamy, která se přizpůsobuje uživateli podle toho, jaký obsah na internetu konzumuje, chtěli jsme podobné možnosti přizpůsobení zavést i do našeho produktu. Z předchozích analýz uživatelského chování jsme věděli, že noví uživatelé mají často problém začít platformu správně používat a často se ztrácejí v základních úkonech, což je pak vede k tomu, že platformu raději opustí. Kdybychom však zvládli analyzovat, na čem konkrétně uživatelé chybují, co jim chybí a jak je postrčit správným směrem, mohli bychom nejen zlepšit uživatelský zážitek, ale také udělat celý proces „nalodění“ do Integromatu mnohem jednodušší.

Smyslem celého projektu tedy je sestavení interní služby, která se bude starat o sbírání uživatelských událostí napříč celou platformou a zajistí jejich odesílání do služeb třetích stran k další analýze. V této práci se budu zabývat nejenom návrhem samotné architektury této služby, a to jak z pohledu datového modelu, tak z pohledu aplikačního návrhu, ale také řešením efektivního odesílání sesbíraných dat, řešením opětovného odesílání událostí v případě selhání komunikace, řešením monitoringu služby, mapováním událostí na jednotlivé cíle třetích stran, meziprocesovou komunikací v případě clusterových běhů a také návrhem ovládacího API.

Nebudu zde naopak rozebírat přímo analýzu uživatelských událostí a vyvozování důsledků z nich, neb to nebylo mým popisem práce.

Kapitola 1

Cíl práce

Cílem práce je návrh a realizace služby Trackman, která bude sloužit pro akumulaci a distribuci událostí, které jsou produkovány ostatními komponentami platformy Integromat, a k jejich následnému odesílání do analytických nástrojů třetích stran, kde bude docházet k jejich vyhodnocování. Zatímco první myšlenka práce byla, že se služba Trackman využije pouze pro sledování toho, co v platformě Integromat dělají přímo uživatelé, postupem času a vývoje se ukázalo, že své uplatnění zároveň najde i čistě na backendu, kde pomůže s monitorováním událostí, které v aplikačním clusteru nastávají a i tyto události bude přenášet do analytických a logovacích nástrojů. K odesílání bude využívat HTTP REST API zmíněných analytických nástrojů, abstrahované pomocí Integromat modulů – konektorů. Díky tomu bude implicitně podporovat možnost odesílat sesbírané události do všech služeb, pro které v platformě již existuje integrace a zároveň tak bude snadno rozšiřitelná o nové konektory s minimální námahou. Nabídne široké možnosti přizpůsobení formátu odesílaných dat a bude tak univerzálním nástrojem pro sledování dění v platformě.

Kapitola 2
Sledování událostí

Hlavním, a poměrně obecným cílem služby Trackman, je sledování událostí. Pod tím si ovšem každý může představit ledasco. Proto bych rád hned ze začátku práce uvedl tento termín na pravou míru a vysvětlil, jaké události jsou tímto pojmem myšleny a jak s nimi budeme pracovat.

2.1 Událost

Architektura Integromatu je založena na mikroslužbách[1], které spolu vzájemně komunikují pomocí různých komunikačních kanálů, ať už se jedná o databázi[2], frontu zpráv[3] nebo meziprocesovou komunikaci[4]. Tyto služby, spojené dohromady, tvoří jeden celek – platformu Integromat.

Událost[5] může být produkována jak na základě uživatelské interakce s některou z komponent, tak zcela automaticky backendovými komponentami a nebo přímo databázovou úlohou, podle aktuálního stavu aplikace. Událostí tedy může být například informace o kliknutí uživatele na tlačítko pro uložení scénáře v aplikaci, stejně tak dobře ale jako příklad události poslouží například zachycená chyba během běhu scénáře na pozadí, který uživatelem vůbec nebyl vyvolán a nastal zcela automaticky.

Všechny tyto události mají důležitou informační hodnotu, každá však pro jinou skupinu zaměstnanců a s každou z nich musí být naloženo jiným způsobem. Že na backendu dochází často k zachycení určitého druhu chyby je důležité převážně pro vývojáře dané služby, naopak z počtu klikání na různé komponenty v rámci uživatelského rozhraní dovede tým starající se o podporu růstu[6] vyhodnotit slabá nebo problémová místa v rozhraní a následně tyto problémy řešit.

Jakákoliv komponenta Integromatu, která má přístup k databázi, může začít do služby Trackman zasílat události, ke kterým uvnitř ní dochází. To, jaké události služba poskytuje, závisí na jejím vývojáři a daných funkčních požadavcích. Úkolem Trackmana v celém Integromat ekosystému je tyto události přijímat a podle zadané konfigurace odbavovat. U událostí, které se týkají uživatelských akcí, zpravidla dochází k jejich odesílání do nástrojů pro analýzu uživatelského chování, jako jsou například platformy Customer.io nebo Userflow. Naopak u chybových událostí dochází k jejich odesílání do monitorovací služby s využitím konektoru Datadog.

[1] Menší služby, mezi které je rozdělena zodpovědnost a společně tvoří komplexní službu jako celek, rovněž budou uváděny jako *microservices*
[2] Zpravidla PostgreSQL
[3] Zpravidla RabbitMQ
[4] Zajišťuje proprietární služba *Integromat Overseer*
[5] Dále také zmiňována pod anglickým ekvivalentem *event*
[6] Budu označovat také anglickým ekvivalentem *Growth Team*

2.2 Uživatelské události

2.2.1 Přínosy

Rád bych teď přiblížil, proč je vlastně vhodné sledovat uživatelské události, proč jsme se jako firma rozhodli je začít sbírat a čeho se pomocí jejich analýzy dá dosáhnout.

Existuje celá řada ukazatelů, které se dají na webových stránkách sledovat. Jejich vyhodnocování nám může pomoci lépe odhadnout profil každého zákazníka a zároveň odhalit případná slabá místa na našich stránkách, která vedou zákazníky k tomu, aby stránky opustili. [1]

Základním ukazatelem jsou takzvaná sezení[7]. Z jejich počtu jsme schopni sledovat počet uživatelů, kteří aplikaci navštěvují. V momentě, kdy uživatel stránky navštíví poprvé, je mu vygenerováno unikátní SID[8] a to se následně ukládá do cookies uživatelova prohlížeče. V případě, že má uživatel cookies vypnuté, nelze session uložit. Pomocí SID jsme pak schopni spárovat jednotlivé požadavky od toho samého uživatele a sledovat tak jeho aktivitu. Zároveň, pokud se uživatel do naší aplikace přihlásí, jsme schopni spárovat jeho aktuální session s jeho profilem, který u nás již má. Toto je výhodné pro sledování toho, jací uživatelé se vracejí, případně za jakým účelem. [10]

Neméně důležité je také sledování takzvaných **pageviews**. Ty reprezentují celkový počet návštěv stránky, respektive celkový počet požadavků na zobrazení obsahu. Je velice výhodné je sledovat, zejména pak na takzvaných **landing pages**, což jsou jednoúčelové stránky, zaměřené na jedno konkrétní téma. Pomocí pageviews lze tedy sledovat navštěvovanost jednotlivých stránek, což může být bráno také jako indikátor kvality obsahu. Pokud je obsah na stránce relevantní, například na kvalitní stránce s nápovědou, uživatelé se budou často vracet, protože je pro ně obsah stránky užitečný[9]. [4]

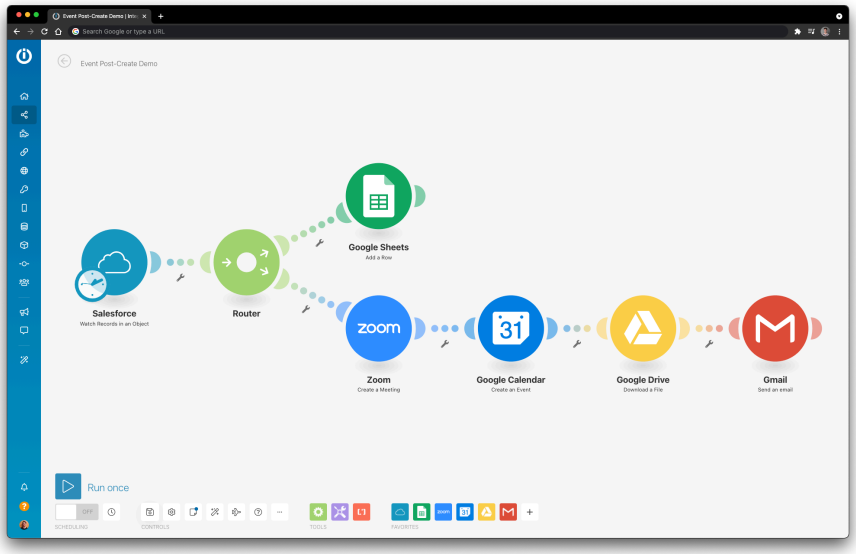
2.2.2 Napojení na službu Trackman

Některé z těchto uživatelských akcí probíhají čistě na frontendu a nelze je tedy zachycovat přímo, jelikož webová stránka samozřejmě nemá přímý přístup k databázi. I takové události je ovšem potřeba sledovat. Jako příklad můžeme použít kliknutí na tlačítko, které pouze ovládá JavaScript v prohlížeči uživatele, ale žádným způsobem po kliknutí neproběhne volání backendu. Konkrétním příkladem v případě Integromatu je ovládání editoru scénářů[10].

Editor scénářů je plně frontendová komponenta aplikace Integromat, která slouží k tvorbě a editaci vlastních integrací a je tak naprosto stěžejní komponentou celé platformy. Zároveň díky tomu, že je postavena na HTML elementu `canvas` nabízí pixel-perfect kontrolu celého prostředí, což má velice pozitivní vliv na celý uživatelský zážitek při používání této komponenty. Součástí Surface jsou také ovládací tlačítka, která se nacházejí v dolní části editoru. Některá z nich, jako například *Spuštění scénáře*, provádějí volání backendového REST API, kde následně může dojít k jejich zaevidování, ovšem jiná, jako například *Zobrazení poznámek*, pouze mění vizuální stránku editoru bez toho, aby prováděla API volání backendu. Dalším příkladem takového čistě frontendového tlačítka může být zavření modálního okna kdekoliv v aplikaci.

Abychom mohli tento typ událostí sledovat, bylo potřeba modifikovat backendové API tak, že jsme do něj přidali otevřený sledovací endpoint, který je provolávatelný AJAXem z frontendu a je chráněn běžnou formou autorizace. Když tedy dojde k nějaké události, která by sama o sobě nezpůsobila volání backendu, nicméně Growth Team by tuto událost potřeboval sledovat, napojíme na ni na frontendu asynchronní volání tohoto endpointu a tak jsme schopni ji na pozadí zaevidovat a odeslat Trackmanovi do fronty událostí k odeslání.

[7] Neboli *sessions*
[8] Session ID
[9] A případně mají odkaz uložený v záložkách prohlížeče
[10] Komponenta *Integromat Surface*



Obrázek 2.1 Editor scénářů *Surface*

```
PERFORM public.trace('scenario_saved',
        jsonb_build_object(
                'user_id', _user_id,
                'user_email', (SELECT u.email FROM users.user_get(_user_id) u),
                'organization_id', _r.company_id,
                'scenario_saved_at', current_timestamp,
                'scenario_saved_name', _r.name,
                'scenario_saved_id', _id,
                'scenario_saved_schedule', _scheduling->>'type',
                'scenario_active', _r.islinked
                ));
```

Výpis kódu 2.1 Ukázka volání procedury `public.trace`

Většina uživatelských událostí ovšem sama z podstaty události způsobuje volání backendu, jedná se o jakékoliv uložení, potvrzení, smazání, vyplnění formuláře, zkrátka o naprostou většinu událostí, které může uživatel v rámci platformy Integromat provést. U těchto událostí je napojení na službu Trackman podstatně snazší. Hlavním vstupním bodem do celého odesílacího procesu je trasovací funkce `public.trace()`, kterou si blíže rozebereme později. Integromat obecně stojí na tom, že spousta logiky je uložena přímo v databázových funkcích a procedurách. Do všech ukládacích procedur, které by měly tedy nějakou událost produkovat, stačilo přidat volání této uložené funkce s patřičnými parametry, které budou vysvětleny později, a napojení bylo dokončeno.

2.3 Backendové události

Druhou skupinou událostí, kterou je Trackman připraven sledovat, jsou backendové události, které nebyly vyvolány žádnou uživatelskou akcí. Způsob jejich sledování nyní rozeberu podrobněji.

Díky monitorovacím nástrojům, kterými v současné době jsou interní nástroj Overseer a ex-

terní, komerční platforma Datadog, máme i bez využití služby Trackman poměrně detailní přehled o tom, k čemu uvnitř služeb dochází, zejména díky aplikačním logům, které jsme schopni sbírat a následně v nich vyhledávat, případně díky metrikám a interaktivním tabulím, které sestavujeme na platformě Datadog a díky tomu v téměř reálném čase v grafech vidíme, co se ve které z komponent odehrává.

Jsou ovšem místa, kde nám ani jedna ze zmíněných možností nepomůže. Existují kritické procesy, u kterých je potřeba na vzniklou chybu reagovat okamžitě. Ačkoliv se k architektuře služby Trackman dostanu až v následujících kapitolách, již teď nastíním, že spousta logiky je prováděna nikoliv v aplikační, nýbrž v databázové vrstvě. Toho jsme dosáhli s využitím RDBMS PostgreSQL[9] a jeho funkcionality uložených funkcí a procedur, která je ještě obohacena skvěle fungujícím modelem přidělování uživatelských práv jednotlivým službám.

Uvnitř těchto databázových funkcí ovšem nemáme k dispozici volání pro službu Overseer, které by muselo být provedeno z aplikační úrovně. Oznámení o kritické chybě tedy nemůžeme provést touto cestou. Samozřejmě, že bychom mohli oznámení o chybě poslat do databázového logu, což se také děje, ale vyhodnocování logu je pomalé a založené na textové podobnosti, tedy ne vždy spolehlivé. Zde přesně přichází využití služby Trackman.

V případě, že v tomto backendovém procesu na úrovni databáze zjistíme kritickou chybu, kromě zalogování zároveň zavoláme trasovací funkci Trackmana. Trackman do své fronty obdrží novou událost, kterou během několika vteřin vyhodnotí a odešle do monitorovacího nástroje. Jelikož se nejedná o textový log, ale o objekt obsahující strojově čitelná data, můžou na tuto událost být v monitorovací službě napojeny další akce, například rozeslání notifikací.

Trackman tedy najde své uplatnění i při monitoringu těchto částí aplikačního clusteru a stává se tak důležitou komponentou, která pomáhá se zajištěním stability platformy.

Kapitola 3
Analýza a návrh

První praktickou částí mé práce byla analýza problému sledování uživatelů a návrh architektury aplikace, která tento problém zvládne vyřešit. Nejprve projdu body, na kterých jsem architekturu začínal stavět a následně budeme postupovat až k funkčnímu prototypu.

3.1 Funkční požadavky

3.1.1 FP1 – Univerzální vstupní rozhraní

Služba Trackman musí poskytnout univerzální vstupní rozhraní pro všechny ostatní komponenty platformy Integromat, které běží ve stejném clusteru. To znamená, že pro jakoukoliv komponentu, která bude chtít do systému začít posílat k jakým událostem v ní dochází, bude existovat jednotný postup, jak tohoto cíle dosáhnout.

3.1.2 FP2 – Využije konektory Integromatu

Trackman má být připraven na odesílání událostí na různé cíle, avšak vždy pomocí jejich HTTP REST API. Aby bylo co nejjednodušší přidat do Trackmana podporu pro nový cíl, tedy pro nové API, bude Trackman k odesílání událostí na služby třetích stran využívat přímo konektory z platformy Integromat. V praxi to znamená, že pro přidání nového cíle do Trackmana musí být na Integromatu vyvinuta integrace s touto službou. Následně bude Trackman schopen používat moduly z této integrace pro odesílání dat. Dojde tím k naprostému odstínění kódu konektorů od kódu aplikace jako takové, což je velice žádoucí a účelné.

3.1.3 FP3 – Umožní definovat podobu dat pro každý cíl

Různé cíle budou potřebovat různé struktury dat, které budou jejich API akceptovat. Trackman umožní pro každý unikátní pár událost--cíl specifikovat podobu těla odesílaného HTTP požadavku. Díky tomu bude možné přesně určit, v jaké podobě se data kam posílají a trefit se tak do vstupních rozhraní API endpointů třetích stran. Ke splnění tohoto funkčního požadavku využije Trackman jazyk IML[1], aby byl datový formát specifikace shodný s konfiguracemi v jiných částech platformy Integromat a pro správce Trackmana bylo tedy jednoduché a intuitivní mapování vyplňovat.

[1] Jazyk *Integromat Markup Language* pro popis datových transformací

3.1.4 FP4 – Dávkové odbavování

Trackman bude přijaté uživatelské události agregovat do fronty událostí k odeslání. Jednou za specifikovaný časový interval, který bude možné definovat v konfiguračním souboru, vybere z fronty podle priority a času přijetí top n událostí a tyto události zpracuje a odešle. Nebude tedy fungovat stoprocentně v reálném čase, ale bude periodicky kontrolovat frontu a z ní bude dávkově události odbavovat.

3.1.5 FP5 – Rychlostní optimalizace mapovacího stromu

Uživatelem definovaná mapování z FP3 – Umožní definovat podobu dat pro každý cíl budou uložena v databázi, ale zároveň budou také načtena a naparsována přímo v paměti. Díky tomu při zpracování události nebude pokaždé docházet k načtení odpovídajícího popisu mapování z databáze, namísto toho se Trackman pouze podívá do své paměti a z ní popis mapování použije. Trackman bude automaticky kontrolovat změny mapovacího stromu v databázi a v případě potřeby sám zařídí obnovení stromu v paměti.

3.1.6 FP6 – Podpora CRONů

Trackman bude podporovat datové CRONy. CRONy v případě Trackmana definujeme jako dávkové úlohy, operující nad velkým počtem řádků v databázi, typicky se bude jednat o periodicky prováděné agregace uživatelských dat. Tyto CRONy, ačkoliv nejsou přímo způsobeny nějakou uživatelskou akcí, budou své výstupy službám třetích stran poskytovat skrze službu Trackman stejným způsobem, jako tomu tak je u událostí přímo vygenerovaných uživatelem. CRONy budou napsány v podobě SQL procedur a budou volány periodicky v časovém intervalu daném konfigurací služby. Díky schopnosti Trackmana nastavovat pro každou kombinaci událost--cíl unikání mapování budou moci být výsledky agregačních CRONů odesílány na jiné cíle, respektive na jiné API endpointy, než běžné uživatelské události.

3.1.7 FP7 – Interní monitoring

Trackman využije komponentu Overseer pro reportování svého aktuálního stavu. Bude odesílat informace o tom, jaké události právě zpracoval, kolik jich bylo, jestli prošla jejich validace a také jak dopadlo jejich odeslání na cílovou destinaci. Tato data následně knihovna předá připojené monitorovací službě.

3.1.8 FP8 – Podrobné debugovací výstupy

Trackman bude obsahovat možnosti zapnutí podrobného výpisu prováděných akcí. S využitím knihovny `@integromat/debug`[2] budou jednotlivé debugovací výstupy rozděleny do různých jmenných prostorů tak, aby se dala vždy aktivovat jenom část potřebná pro ladění daného problému a nedocházelo k zahlcování konzole nepotřebnými výstupy.

3.1.9 FP9 – Validace těl událostí

V případě, že bude možnost validace těl zapnutá, respektive bude pro danou událost poskytnuta validační šablona, Trackman bude automaticky validovat podoby obsahů událostí na jeho vstupu. V případě, že zpráva nebude vyhovovat šabloně, vypíše Trackman chybu a nebude nevalidní událost dále zpracovávat. Tímto způsobem bude zajištěno, že v případě, že začnou z nějaké

[2]Knihovna *Integromat Debug* zajišťující standardizaci a serializaci zpráv do logu

z komponent přicházet nevalidní události, nedojde k pokažení dat na cílových destinacích, jelikož Trackman nevalidní data nepropustí.

3.1.10 FP10 – Retenční mechanizmus

Může se stát, že z různých důvodů neproběhne správně odeslání události na cílovou destinaci. Vzhledem k tomu, že Trackman používá Integromat moduly, které zpravidla komunikují s cílovou stranou pomocí HTTP protokolu, může dojít hned k několika možným problémům, jak na straně klienta, tak na straně serveru. V případě, že taková situace nastane, Trackman tuto chybovou situaci detekuje a odloží odeslání zprávy. Toto odložení několikrát zopakuje a pokaždé se po daném časovém intervalu pokusí zprávu, která chybovala, odeslat znovu. Při každém neúspěšném pokusu přitom ohlásí do konzole, že k takové situaci došlo. Teprve po několika neúspěšných odesláních Trackman událost zahodí.

3.1.11 FP11 – RateLimit optimalizace

Jak bylo již zmíněno v FP10 – Retenční mechanizmus, Trackman komunikuje ve výsledku s využitím HTTP protokolu. Jedním z chybových stavů tohoto protokolu je stavový kód 429, který znamená "Příliš mnoho požadavků najednou", je také známý jako "Rate Limit". Odesílací mechanizmus v Trackmanovi zvládne zajistit, že po obdržení tohoto stavového kódu dojde k pozastavení odesílání na dobu, která bude definovatelná pro každý jeden cíl zvlášť. Po vypršení tohoto timeoutu bude Trackman pokračovat v odesílání dat na zadaný cíl. Díky retenčnímu mechanismu dojde také ke znovuodeslání zprávy, která původně chybu 429 vyvolala.

3.1.12 FP12 – Clusterový režim

Trackman bude připraven na běh v clusteru, tedy na běh více instancí služby Trackman vedle sebe, paralelně, na stejném prostředí. Nesmí docházet k tomu, že by vícero instancí odeslalo stejnou zprávu dvakrát, obecně, nesmí dojít k tomu, že by dvě instance prováděly tentýž úkon. Úkonem nemusí být nutně pouze odeslání události na cíl. Zároveň bude Trackman podporovat sdílení informací mezi běžícími instancemi a jejich vzájemnou komunikaci. K dosažení tohoto cíle využije knihovnu Overseer.

3.1.13 FP13 – Webhooky

Kromě toho, že Trackman bude umět přijímat události standardní cestou, z fronty v databázi, bude také podporovat možnost přijímání událostí z externích zdrojů s využitím HTTP Webhooků. Každý webhook bude mít unikátní URL adresu, na kterou bude možné pomocí HTTP volání zaslat data. Trackman data přijatá na této adrese zpracuje, zvaliduje a následně je zařadí mezi ostatní události do fronty k odeslání. Tyto vstupní endpointy bude možné rovněž zabezpečit pomocí validací hlaviček. Validace bude specifikována s využitím IML.

3.2 Technologie a architektura

V této sekci přiblížím různé technologie, principy a vzory, které jsem se rozhodl v práci využít pro splnění zadaných funkčních požadavků.

3.2.1 Integromat

Integromat je cloudová integrační platforma (iPaaS[3]), která dovoluje svým uživatelům automatizovat rutinní procesy a integrovat širokou paletu aplikací mezi sebou, a tím zákazníkům šetřit nejen čas, ale také peníze. V současné době nabízí již přes 700 různých konektorů ke cloudovým službám všech kategorií, například jde o účetní software, skladové systémy, CRM[4] systémy, e-shopové aplikace, mailingové služby a mnoho dalších služeb. Rovněž nabízí univerzální moduly pro připojení k nejpopulárnějším druhům databází, moduly pro transformace datových struktur ve formátu JSON[5] a XML[6] a v neposlední řadě také generický HTTP modul, který umožňuje uživateli odeslat HTTP požadavek na jakoukoliv adresu v Internetu a následně v Integromatu zpracovat odpověď.

Aplikace jsou v Integromatu dostupné v podobě modulů, které mezi sebou poté uživatel provazuje ve vizuálním editoru. Vzniklou integraci označujeme jako scénář. Každý modul zpravidla reprezentuje abstrakci HTTP REST[7] API endpointu dané služby[8]. Vývojář aplikace v Integromatu tedy nastuduje dokumentaci služby třetí strany, která se má do systému přidat, pomocí interní platformy **Apps** naprogramuje sadu modulů a aplikace je poté zpřístupněna zákazníkům. Ti ve scénářích už nemusí řešit formát HTTP požadavku, formát autorizace, názvy endpointů a tak dále, ale zkrátka vloží do svého scénáře požadovaný modul a namapují do něj výstupy modulů předchozích.

Integromat nabízí mnoho pokročilých funkcionalit, které umožňují stavět skutečně robustní a univerzální integrace. Umožňuje scénáře rozdělovat do vícero větví pomocí modulů zvaných Routery, umožňuje následně filtrovat jaká data jsou předávána do té které větve, umožňuje spouštět scénáře externí událostí pomocí Webhooků, které čekají na příchozí HTTP požadavek a následně instantně spustí scénář, zpracují data a odpoví, nabízí také interní datové úložiště pro uživatelská data[9] a mnoho dalšího. Jedná se o pokrokovou, nadčasovou službu, jejíž uživatelská základna roste každým dnem a v posledních měsících se dostává i do povědomí v korporátním segmentu.

Při tvorbě modulů platforma Apps abstrahuje kód v jazyce JavaScript. Pro vývojáře spočívá programování nového modulu v sestavení direktiv v jazyce IML, které jsou následně při kompilaci aplikace převedeny na skutečný JavaScriptový kód. Hotový modul je tedy k dispozici jako JavaScriptová třída, kterou lze naimportovat v jiném místě v kódu, což se zpravidla děje při spouštění scénáře. Zároveň ovšem právě této vlastnosti využívá Trackman. Podle zadání má Trackman umět využít moduly aplikace Integromat pro odesílání dat. Když Trackman odesílá nějakou událost na třetí stranu, naimportuje dynamicky modul dané služby a obdobně, jako kdyby se modul spouštěl ve scénáři, modul spustí a jako vstup mu podstrčí právě odesílanou událost. Tímto způsobem tedy Trackman úplně abstraktně umí spustit **jakýkoliv** modul, který je v Integromatu dostupný, což zjednodušeně znamená, že aniž by se musela programovat vrstva kompatibility, Trackman sám od sebe umí odesílat data do více než 700 služeb, které jsou na Integromatu dostupné.

3.2.2 PostgreSQL

PostgreSQL je v současnosti jedna z nejrozšířenějších a nejuniverzálnějších objektově-relačních open source SQL databází. Nabízí širokou škálu funkcionality i nad rámec standardního SQL a umožňuje bezpečně a spolehlivě skladovat skutečně velké objemy dat a to i pod velice vysokou

[3] Integration Platform as a Service - Integrační platforma jako služba
[4] Customer Relationship Management - software pro správu vztahů se zákazníky
[5] JavaScript Object Notation - formát serializace dat vycházející z jazyka JavaScript
[6] Extensible Markup Language - metajazyk pro popis dat
[7] Ne nutně
[8] Samozřejmě kromě speciálních modulů, jako jsou například databázové konektory
[9] V platformě známé jako Data Stores

```
{
    "url": "/courses/{{parameters.course}}/exams",
    "qs": {
        "limit": 50,
        "sem": "{{parameters.semester}}"
    },
    "method": "GET",
    "response": {
        "output": "{{parseExam(item)}}",
        "iterate": {
            "container": "{{body.atom:feed.atom:entry}}",
            "condition": "{{if(body.atom:feed.atom:entry,true,false)}}"
        }
    },
    "pagination": {
        "qs": {
            "offset": "{{(pagination.page - 1) * 50}}"
        }
    }
}
```

Výpis kódu 3.1 Ukázka specifikace API volání KOS API v jazyce IML

zátěží. Jelikož i ostatní microservices v rámci platformy využívají právě tento RDBMS, rozhodl jsem se pro zajištění jednoduché kompatibility jít v projektu Trackman stejnou cestou.

Klíčovou funkcionalitou je podpora uložených procedur v jazyce PL/pgSQL, kterou Trackman hojně využívá. Tento koncept přináší skvělou možnost, jak některé části logiky zachovat na straně databázového stroje bez nutnosti data vytahovat do aplikační vrstvy, tam je zpracovat a opětovně vrátit do databáze. Kromě jazyka PL/pgSQL umožňuje PostgreSQL psát uložené procedury také například v jazycích Perl, Python nebo C. Existuje dokonce rozšíření PLV8, které nabízí psaní uložených procedur v jazyce JavaScript, toto řešení ovšem nepraktikujeme.[6]

Další zajímavou funkcionalitou je formát JSONB. Jedná se o binární formát, který umožňuje ukládat JSON objekty, ale oproti prostému JSONu, který je v podstatě textovým formátem, nabízí mnohem větší rychlost přístupu k datům, dokonce umožňuje indexování podle zanořených klíčů a zejména při práci s JSONy na straně uložených procedur přímo v databázi je využití formátu JSONB velkou rychlostní výhodou. Tato optimalizace ovšem není zadarmo, nutno říci, že JSONB potřebuje pro uložení stejného datového obsahu více místa na disku. U databáze Trackmana se ovšem neočekává, že by přílišně rostla, takže tam tuto nepříjemnost můžeme zanedbat.

PostgreSQL také nabízí vestavěnou podporu pro replikaci dat. Této využíváme především k tomu, že pro operace, které nepotřebují do databáze zapisovat, ale pouze číst již uložená data, máme vyhrazenou speciální read-only repliku dat. Při spouštění CRON procedur Trackman používá právě tuto repliku, aby velkými dotazy agregujícími uživatelská data nezatěžoval hlavní read-write stroj.

Trackman je navržen tak, aby podporoval PostgreSQL od verze `9.6`, jelikož v době inicializace projektu byla právě tato verze nasazena na produkčním prostředí. Nyní již Trackman samozřejmě běží bez problémů i s verzí `13`.

3.2.3 REST

REST je vedle XML-RPC nebo SOAP dalším architektonickým vzorem pro návrh API. Zatímco další dva zmíněné vzory popisují co se má s daty na serveru stát[10], REST je datově orientovaný a spočívá v definici jednotlivých **zdrojů**, se kterými se následně pracuje.[5] Jelikož Trackman a Integromat obecně používají protokol HTTP ke komunikaci se třetími stranami, budu se zde věnovat analýze právě HTTP REST API.

Základními požadavky REST architektury jsou:

- **Separace zodpovědnosti**
  Klient je ten, kdo se ptá na data a posílá požadavky, server je zpracovává a odpovídá na ně.
- **Vrstvená architektura**
  Z pohledu klienta je odeslání požadavku zpravidla zavolání jednoho určeného endpointu. Klient neřeší, zda je na straně serveru nastavený load-balancing, proxy, přes kolik serverů ve finále požadavek protéká, ale zkrátka zavolá adresu pro získání zdroje podle předem dané specifikace a serverová strana požadavek zpracuje.
- **Uniformní rozhraní**
  Rozhraní založené na zdrojích, se kterými se manipuluje pomocí volání metod nad nimi. K tomuto jsou hodně využívány různé HTTP metody, které pak specifikují, jaká akce se má se zdrojem stát. S tímto se pojí zároveň koncept HATEOAS[11], kdy jedna entita obsahuje odkazy na další entity, které jsou s ní spojené a pomocí těchto odkazů se pak dá jednoduše procházet celý graf závislostí. Například pokud pracujeme s objednávkou, bude obsahovat odkazy na všechny zakoupené položky, každá zakoupená položka pak obsahuje odkaz na produkt a zároveň zpět na objednávku, do které patří.
- **Bezstavovost**
  Komunikace mezi klientem a serverem je bezstavová, tedy server si nepamatuje žádný kontext mezi dvěma různými požadavky od stejného klienta. Klient je pasivní, posílá požadavky a dostává na ně odpovědi. Klient si samozřejmě stav udržuje na své straně a podle něj posílá navazující požadavky, ale pro server se každý požadavek jeví jako úplně nový. Neexistuje zde tedy žádná transakčnost, na druhou stranu tento koncept dává prostor pro velice dobrou škálovatelnost, protože v případě, že server běží ve vícero instancích, není potřeba rozlišovat, na kterou instanci požadavek dorazil, protože si instance nic nepamatují a každá tak vrátí za všech okolností stejný výsledek.
- **Cachovatelnost**
  Tento koncept je spojený s bezstavovostí. Vzhledem k tomu, že většina REST požadavků se zpravidla soustředí na získání dat ze zdroje, je možné data cachovat a zajistit tak větší propustnost systému. V případě souběhu mnoha požadavků na přístup k tomu samému zdroji je pak daleko efektivnější mít cache[12] v paměti před databází[13].

Zdroje jsou v REST identifikovány jednoznačně a zpravidla hierarchicky. Pokud máme například API, které poskytuje informace o knihách v knihovně a chceme u nějaké dané knihy získat seznam autorů, může endpoint pro získání dat vypadat třeba takto: `/books/<bookId>/authors`. Jasně tedy hned z adresy poznáme, že nás zajímá kniha se zadaným ID a pro tuto knihu chceme získat seznam autorů.

Pro manipulaci se zdroji se využívá několik různých HTTP metod.

Odpovědi od serveru v případě HTTP REST API využívají stavových HTTP kódů.

[10] Například XML-RPC v podstatě přesně říká jakou proceduru s jakými vstupními daty spustit
[11] Hypermedia As The Engine Of Application State
[12] I kdyby jen několik sekund
[13] Lze použít přímo v aplikaci, nebo zajistit například službou Redis

Tabulka 3.1 Tabulka HTTP metod ve vztahu k REST

HTTP Metoda	Konkrétní Instance	Kolekce
GET	Získá detail instance	Získá seznam instancí v kolekci
POST	Provedení generické akce	Vytvoří nový prvek v kolekci
PUT	Vytvoření nebo náhrada	Náhrada celé kolekce
DELETE	Smazání	Smazání celé kolekce
PATCH	Částečná úprava	*Nemá standardizované chování*

- **2xx**
  Používá se pro oznámení úspěchu. Nejčastěji se setkáme s kódem `200 OK`, což je zkrátka oznámení úspěšně provedeného požadavku, některé implementace ovšem vrací i detailnější kódy, například `201 Created` oznamující založení nové entity, případně `204 No Content`, který značí, že v těle odpovědi nemá klient hledat žádná data. Zpracování této odpovědi je tak optimalizovanější, protože klient[14] rovnou přeskočí krok zpracování těla odpovědi.

- **3xx**
  Kódy pro řízení požadavku. V této řadě najdeme kódy pro přesměrování žádosti, například `301 Moved Permanently` nebo `302 Found`. Správně implementovaný klient by si měl odpověď `301` zapamatovat a při dalším volání této adresy automaticky sám rovnou poslat požadavek na cílovou lokaci, která byla oznámena v první `301` odpovědi.

- **4xx**
  Značí chybu na straně klienta. Zpravidla jsou vráceny, když je požadavek formulován nesprávně, ale také řeší například neexistenci cesty, neautorizovanost požadavku nebo třeba přehlcení požadavky od daného klienta[15].

- **5xx**
  Značí chybu na straně serveru. Klient se tímto kódem dozvídá, že se serverem něco není v pořádku a měl by opakovat akci později, avšak opravu musí zajistit správce serveru nebo jeho vývojář.

3.2.4 Microservices

Microservices jsou jedním z návrhových vzorů pro propojení vícero aplikačních komponent dohromady. Při použití microservices je zodpovědnost rozdělena do více služeb, z nichž každá zajišťuje jednu konkrétní činnost a všechny dohromady poté tvoří jeden celek – v případě Trackmana jde o platformu Integromat.

Tento koncept je skvělý z hlediska škálování. Kromě ojedinělých případů by každá ze služeb měla být spustitelná vícekrát vedle sebe a díky tomu je pak velice snadné škálovat výkon takzvaně do šířky, tedy v případě, že výkon jednoho serveru již nestačí, přikoupí se druhý server a služba se na něm pustí znovu. Pomocí loadbalancingu pak zajistíme, že se zátěž rovnoměrně rozděluje mezi tyto dvě instance.

Tento koncept je také důležitý pro zajištění plynulých restartů a aktualizací. Restart je potřeba například ve chvíli, kdy došlo ke změně konfigurace služby, nebo je k dispozici nová verze aplikačního kontejneru. Díky tomu, že služby běží paralelně vedle sebe, lze nejdříve ukončit pouze jednu z nich, přičemž druhá stále zpracovává příchozí události, a jakmile se první instance aktualizuje a znovu spustí, provede se aktualizace instance druhé. Tímto způsobem je pak možné se zachováním stoprocentní dostupnosti vyrestartovat celý cluster.

[14] Pokud je správně naimplementovaný
[15] Také známé jako Rate Limit

Zmíněná zástupnost služeb se hodí také v případě neošetřené chyby a pádu aplikace. Load balancer, který se stará o rozložení zátěže, automaticky vyhodnotí, že některá z běžících instancí má problém, a přestane na tuto instanci posílat požadavky. V momentě, kdy se instance vyrestartuje[16], load balancer pozná, že je opět dostupná a požadavky začne opět posílat.

Jednotlivé služby mezi sebou komunikují pomocí předem definovaného rozhraní. Může se jednat nejenom o HTTP REST API, komunikace může probíhat také na úrovni databáze nebo přes frontovací službu[17].

Téměř opakem k architektuře mikroslužeb je architektura monolitická. Ta stojí na konceptu, že celá služba je jako celek zabalena do jedné aplikace. Na tomto konceptu je v dnešní době stále postavena spousta aplikací, především velkých softwarových služeb určených pro korporátní použití. Výhodou a zároveň nevýhodou je, že je všechen kód na jednom místě. Pokud monolitická aplikace překročí určitou mez, stává se obtížnou na správu a zároveň tento koncept není úplně dobře škálovatelný. Aby se možnosti alespoň trochu zlepšily, bývají monolitické aplikace zpravidla rozděleny na různé interní komponenty, které spolu komunikují.[7]

3.3 Terminologie

Napříč celou prací budu pojmenovávat různé části služby Trackman, entity, se kterými pracuje, a odkazovat se na ně. V této kapitole bych rád tuto terminologii rozebral a vysvětil.

3.3.1 Target

Prvním pojmem, který zde představím, je target, neboli cíl. Target reprezentuje cílovou destinaci, na kterou má Trackman odesílat data. V databázi je reprezentován tabulkou `trackman.target`. Jeho definice se skládá především z trojice `app_name`, `app_version` a `module_name`. Tato trojice je totiž v rámci platformy Integromat jednoznačným identifikátorem právě jednoho modulu v jedné aplikaci.

Jak již bylo několikrát zmíněno, Trackman používá přímo Integromat konektory k tomu, aby pomocí nich odesílal data na služby třetích stran. Pro lepší představu uvedu praktický příklad. Jednou ze služeb, kterou Integromat reálně používá pro analýzu a vyhodnocování chování uživatelů, je v současné době platforma Customer.io. Integrace s touto aplikací v Integromatu obsahuje několik modulů, kdy jedním z nich je modul Track a Customer Event.

3.3.2 Event

Dalším, neméně důležitým pojmem, je samotná událost. Události jsou produkovány ostatními mikroslužbami a mohou představovat v podstatě jakoukoliv akci, ke které v rámci platformy dochází. Některé z nich přímo souvisejí s chováním aktuálně přihlášeného uživatele, jako je například akce „Uložení scénáře“ nebo „Spuštění scénáře“, jiné události jsou produkovány zcela automaticky bez uživatelského zásahu. Mezi takové řadíme například „Selhání scénáře na pozadí“.

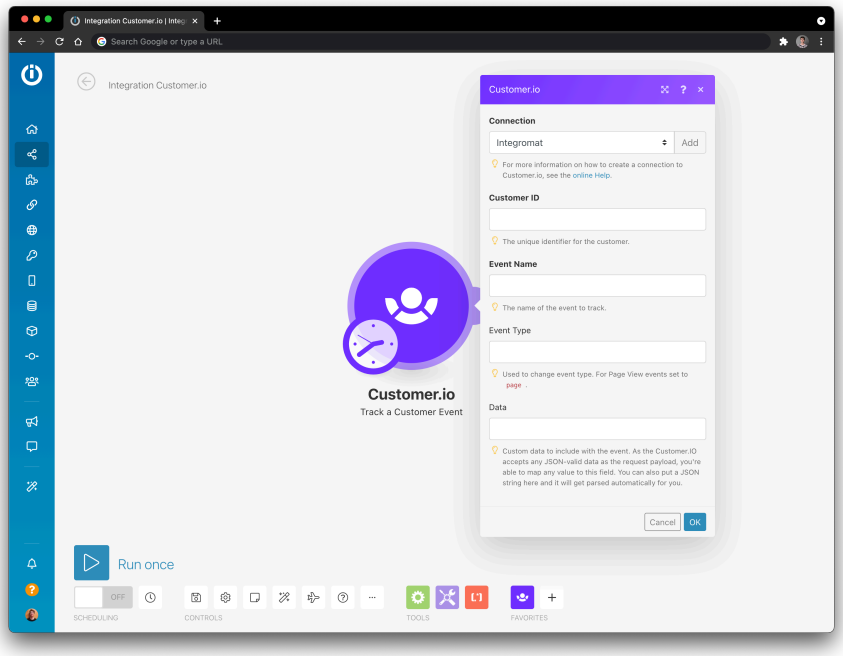
Každý typ akce musí mít unikátní název v rámci celé platformy a právě jednu službu, která tuto akci produkuje. Mikroslužba, která chce akci poskytovat, ji při svém startu registruje do databázového schématu. Pokud je již požadovaný název obsazený, registrace události je zamítnuta a vývojář musí následně zvolit jiný název, pod kterým bude požadovanou akci produkovat. Každá událost je popsána interním formátem Integromatu pro popis datových sad.

Akce je tedy identifikována unikátním názvem a v jejím těle se nalézají další informace. U akcí, které provádí uživatel, zpravidla v jejich těle najdeme identifikátor uživatele a organizace, pod kterou danou akci provedl.

[16] Tyto automatické restarty zpravidla řeší orchestrátor, například Kubernetes nebo Overseer
[17] RabbitMQ, Redis, AWS SQS

```
{
    "provider": "web-api",
    "events": [
        {
            "name": "team_created",
            "label": "Team Created",
            "description": "New team (organization) created by a user",
            "payload": [
                    {
                        "name": "user_id",
                        "type": "uinteger",
                        "label": "User ID"
                    },
                    {
                        "name": "user_email",
                        "type": "email",
                        "label": "User Email"
                    },
                    {
                        "name": "organization_id",
                        "type": "uinteger",
                        "label": "Organization ID"
                    },
                    {
                        "name": "team_created_at",
                        "type": "date",
                        "label": "Team Created At"
                    },
                    {
                        "name": "team_name",
                        "type": "text",
                        "label": "Team Name"
                    },
                    {
                        "name": "team_id",
                        "type": "uinteger",
                        "label": "Team ID"
                    }
            ]
        }
    ]
}
```

Výpis kódu 3.2 Ukázka popisu datové struktury události



Obrázek 3.1 Rozhraní modulu *Track a Customer Event*

3.3.3 Mapping

Mapping, neboli Mapování, je stěžejní pojem celé aplikace Trackman, na kterém je v podstatě postavena celá myšlenka aplikace, a který zajišťuje tolik potřebnou vysokou míru univerzálnosti. Jedná se o způsob, jakým se jednotlivé události mapují na dané cíle. Hlavní složkou mappingu je zápis v jazyce IML, který v zásadě popisuje datovou transformaci, kterou má Trackman provést.

Různé události mají různé formáty těl, tedy obsahují různé údaje, tedy různé proměnné. Cíle, které jsou definovány výše, naopak poskytují abstraktní vrstvu pro práci s REST API třetí strany a jejich vstupní rozhraní je tedy proměnlivé podle toho, v jakém formátu třetí strana data přijímá.

Úkolem mapování je zajistit, že data z události budou korektně transformována to formátu, v jakém je přijímá zadaný cíl. Právě toho je dosaženo pomocí provedení transformace podle IML popisu.[8]

Mapping je tedy definován vždy pro kombinaci event-target a pro každou takovou dvojici v rámci aplikace Trackman může existovat pouze jediný mapping. Dá se tedy říci, a v datovém modelu je to tak skutečně namodelováno, že kombinace událost-cíl je primárním klíčem pro každý mapping.

Dále se u mappingu dají nastavovat vedlejší možnosti, jako je například zapnutí debugovacích výstupů či dočasná deaktivace. Zapnutí debugovacích výstupů se projeví tak, že Trackman při každé aplikaci IML transformace vypíše do systémové konzole zdrojová data, používanou IML šablonu a výsledek transformace. Správce mapování má tedy jednoduchou možnost v konzoli kontrolovat, zda se události mapují na cíle tak, jak je požadováno.

3.3.4 CRON

Události pokrývají většinu informací, které byly zadány ve vstupních požadavcích ke sledování. Existuje ovšem skupina údajů, kterou událostmi sledovat nelze, neboť by to bylo komplikované

```
{
  "id": "{{payload.organization_id}}:{{payload.user_id}}",
  "data": {
    "user_id": "{{payload.user_id}}",
    "tracked_at": "{{event.tracked_at}}",
    "scenario_id": "{{payload.scenario_id}}",
    "organization_id": "{{payload.organization_id}}",
    "scenario_executed_manually_at": "{{payload.scenario_executed_at}}"
  },
  "name": "{{event.label}}"
}
```

Výpis kódu 3.3 Ukázka transformace v jazyce IML

nebo to například skrývalo problémy se synchronizací událostí. Příkladem takového údaje je například „počet aktivních scénářů daného uživatele“. Ačkoliv má Trackman k dispozici událost „Scénář Aktivován“, z podstaty návrhu se zároveň jedná o **bezstavovou mikroslužbu**, která navíc může běžet paralelně. Tento přístup byl zvolen pro skvělé možnosti škálování do šířky, které z něj vyplývají. Na druhou stranu ovšem zavírá možnost, že by Trackman mohl například sám počítat, kolik scénářů má uživatel aktuálně aktivní podle toho, kolik aktivačních a deaktivačních událostí se k němu dostane.

Z toho důvodu bylo potřeba do Trackman služby implementovat způsob, jakým sbírat tyto agregované údaje a rovněž je zasílat podle potřeby do analytických služeb třetích stran. Řešením tohoto problému je implementace takzvaných CRONů. CRONem v kontextu Trackmana uvažujeme databázovou funkci, která je spouštěna jednou za určitý časový interval, v našich podmínkách se zpravidla jedná o 24 hodin. Funkce při jejím zavolání vygeneruje do fronty seznam událostí se speciálním prefixem `cron_`. Tento prefix indikuje, že se právě jedná o událost vytvořenou touto „cronovou funkcí“.

Trackman následně tyto události zpracovává stejným způsobem, jako by se jednalo o živě generované události. Pro tyto události je následně specifikován takový mapping, aby korektně došlo k aktualizaci agregovaných dat na serverech třetích stran.

3.3.5 Fronta událostí

Fronta událostí je reprezentována speciální tabulkou v databázi a obsahuje události, které byly ostatními mikroslužbami odeslány ke zpracování. Když jakákoliv komponenta provede volání trasovací funkce `public.trace()`, dojde k zafrontování události právě do této tabulky, respektive do této fronty.

Trackman události ve frontě událostí odbavuje v pořadí, ve kterém byly trasovány, tedy zachovává pořadí událostí. Událost může mít ve frontě speciální příznak `skipped_until`. Tento příznak říká, do jakého času nemá být tato událost odbavena. Jak bude blíže rozebráno později, tento příznak se používá ve chvíli, kdy dojde k selhání při odesílání události. Aby nedošlo ke ztrátě dat, je odeslání události odloženo o zadaný časový interval, než se Trackman pokusí jej provést znovu. Aby při čtení fronty Trackman věděl, které události má přeskakovat, používá se právě tento popsaný příznak, který je uložen v samostatném sloupci tabulky.

Jelikož Trackman je spouštěn v několika paralelních instancích, je zároveň hlídáno, že nedojde k odeslání jedné události dvěma různými instancemi. Toho jsem docílil využitím možností zamykání řádků v rámci transakcí v jazyce SQL a detailnější princip bude rovněž rozebrán v následujících kapitolách.

3.3.6 Fronta úkolů

Druhou frontou, kterou aplikace Trackman odbavuje, je fronta úkolů. Fronta úkolů má při odbavování přednost před frontou událostí. Úkolem v této frontě může být například „Občerstvit cache“ nebo „Restartovat pracovní proces“. Pokud je fronta úkolů prázdná, automaticky dojde ke zpracování úkolu „Odbavit frontu událostí“. Skrze frontu úkolů lze tedy do jisté míry ovládat chování celé aplikace.

Kapitola 4
Realizace

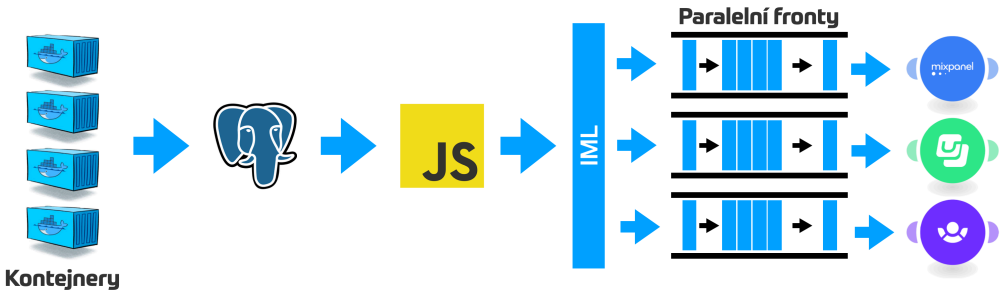
V této kapitole se konečně dostáváme k hlavní části mojí práce, tedy k samotné implementaci dříve popsané služby Trackman. Postupně zde budou rozebrány jednotlivé složky práce, jak na databázové, tak na aplikační vrstvě.

4.1 Rozdělení zodpovědnosti

Klíčovým bodem celé implementace je správné rozdělení odpovědností mezi jednotlivé vrstvy aplikace. Trackman je vyvíjen nad databázovým strojem PostgreSQL ve skriptovacím jazyce JavaScript spouštěném přes NodeJS. Na rozdíl od mnohých řešení, která pro komunikaci s databází využívají dostupné ORM frameworky, Trackman, stejně tak jako drtivá většina ostatních komponent platformy Integromat, jde směrem uložených databázových procedur. Na tomto přístupu si v Integromatu zakládáme neb nám poskytuje mnoho výhod právě oproti využívání libovolného ORM frameworku. Službu Trackman lze tedy primárně rozdělit na dvě vrstvy. Vrstva aplikační a vrstva databázová.

4.2 Databázová vrstva

Databázová vrstva tedy obsahuje nejenom vlastní tabulky pro ukládání dat, ale také spoustu logiky, která je zde uložena ve formě uložených procedur v jazyce SQL, případně PLpgSQL. Zodpovědností této vrstvy je tedy bezchybná reprezentace dat, uchování jednotlivých eventů, targetů, mappingů a dalších řídicích informací služby Trackman a zároveň manipulace s nimi. Na



Obrázek 4.1 High-level diagram fungování aplikace

jednu stranu se zde možná poněkud dostáváme do takzvaného vendor locku[1], tedy jakéhosi zámku v rámci používané technologie, neboť přenášet kromě vlastního datového modelu i procedury s ním spojené a přepisovat je do jazyka jiného databázového stroje by bylo nepochybně náročné, na druhou stranu, vzhledem k tomu, že rozhodně neplánujeme databázový stroj měnit, je pro nás tento zámek přípustný.

Naopak kombinace tabulek i procedur uložených v databázi pomáhá lepší čitelnosti kódu a lépe odděluje aplikační logiku od datové logiky. Aplikační logika totiž ve spoustě případů pouze zvaliduje vstupní parametry a odchytí tak základní validační chyby, jako jsou nevyplněná vstupní pole či chybné datové typy. Pak ovšem dochází k provolání databázové procedury, kde je uložená vlastní výkonná logika. Vzhledem k tomu, že je procedura přímo v databázi, nemusí se doptávat na žádná další data a vše probíhá plynule v rámci jednoho volání procedury a v rámci jedné transakce.

V databázových procedurách je tedy myšleno na hlídání závislostí, hlídají se rovněž různá integritní omezení, dochází k pokročilejší validaci. Na rozdíl od ORM, kde dochází ke generování SQL kódu za pomocí ORM frameworku, při tomto přístupu je každý řádek kódu pevně v rukách vývojáře dané aplikace. Na druhou stranu to znamená, že tento přístup vyžaduje větší expertnost v rámci SQL jazyka, protože jsou všechny procedury definovány přímo vývojářem a nikoliv generátorem. Právě možnost mít vše plně pod kontrolou do nejmenšího detailu je pro nás ovšem v Integromatu klíčová, což tento přístup jenom podporuje.

Pro připojení k databázi je v aplikaci Trackman využit ovladač `@integromat/pg`[2].

Největším přínosem použití proprietárního, interního řešení oproti čistému ovladači, který je volně k dispozici, je bezpochyby jeho rozšíření Evoman. Tato nástavba je totiž schopná sama automaticky rozpoznávat změny v databázových procedurách, které jsou verzovány společně se zbytkem aplikačního kódu ve společném git repozitáři. Procedury tak nejsou uložené pouze na databázovém serveru, ale zároveň v podobě souborů přímo v projektu. Velkou výhodou tohoto řešení je, že databázový ovladač si při každém připojení k databázi automaticky zkontroluje, zdali se shodují procedury v projektu s procedurami na serveru a v případě rozdílů provede synchronizaci. Z pohledu vývojáře je poté práce s procedurami skutečně snadná, jelikož celou logiku hlídání změn a synchronizace za něj automaticky a na všech prostředích vyřeší databázový ovladač.

4.2.1 Tabulky

Rád bych se nyní podrobněji věnoval datovému modelu aplikace Trackman a rozebral jednotlivé tabulky, které se v databázi aplikace nacházejí. Jenom podotknu, že všechny tabulky se nacházejí ve společném schématu `trackman` a jsou tak přehledně odděleny od dalších schémat určených pro jiné komponenty.

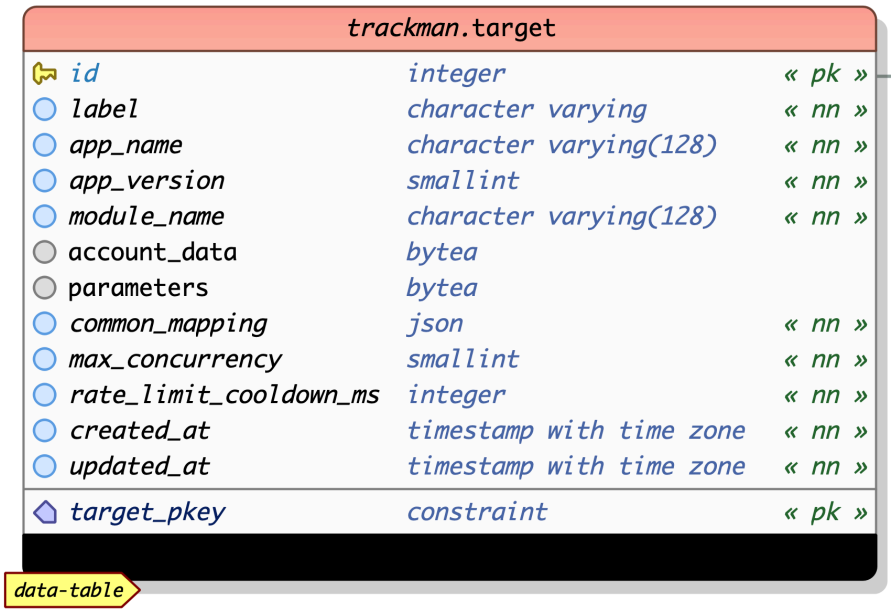
4.2.1.1 Target

Tabulka `trackman.target` uchovává informace o jednotlivých cílech, které jsou definovány pro odesílání událostí. Sloupec `id` je primárním klíčem této tabulky a jedná se o generovanou sekvenci čísel začínající jedničkou pro každý nový řádek v tabulce. Dále obsahuje pole `label`, které dává možnost si zadaný cíl pojmenovat, slouží to pro zlepšení přehlednosti. Dost často se totiž stává, že se do jedné služby data posílají různými způsoby, nebo pod různými účty, a v tu chvíli je potřeba mít možnost lépe odlišit, o jaký cíl se jedná.

Důležitá je trojice polí `app_name`, `app_version` a `module_name`. Tato trojice je jednoznačným identifikátorem modulu v rámci celé platformy Integromat. Jak bylo již několikrát zmíněno, Trackman pro odesílání událostí do služeb třetích stran využívá právě moduly aplikace Integromat, které abstrahují práci s vlastním REST API cílových služeb. Pomocí této trojice tedy

[1] Tedy jsme nuceni používat stále jednu a tu samou službu a její případná výměna je velice nákladná

[2] Upravený databázový ovladač *NodePostgres* pro potřeby Integromatu



Obrázek 4.2 Tabulka `trackman.target`

určíme aplikaci, její verzi a název modulu, který se má pro odesílání dat používat. Způsob, jakým je odeslání provedeno, rozeberu později.

Dále tabulka obsahuje pole `account_data` a `parameters`, které jsou typu `bytea`, který v PostgreSQL databázi představuje binární data. Tato pole jsou šifrovaná na úrovni aplikace a tedy databáze nemůže žádným způsobem číst, co je obsahem těchto polí, a právě proto jsou pole také uložena jako prosté binární buffery. Slouží pro ukládání citlivých dat týkajících se daného cíle, zpravidla pro uložení API klíčů a jiných autorizačních údajů.

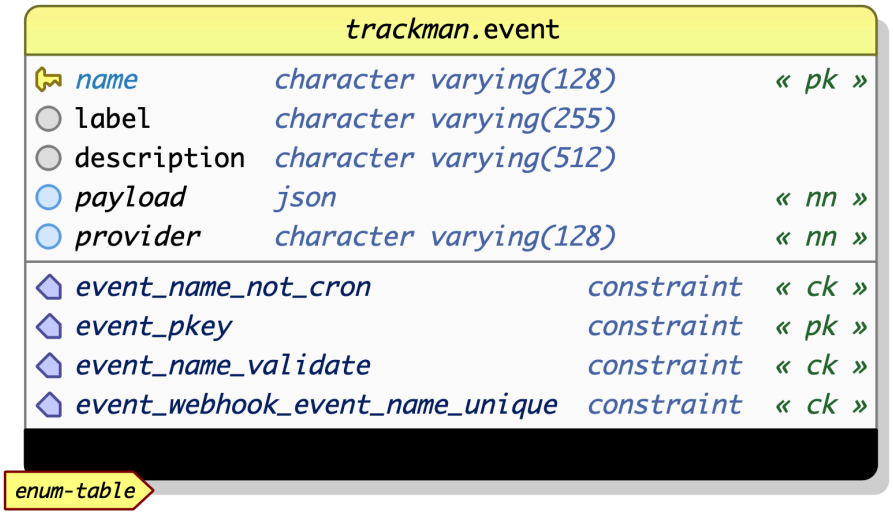
Sloupec `common_mapping` slouží pro uložení specifikace IML transformace, která bude společná pro všechny mappingy, které využívají tento target. Tento způsob uložení je výhodný například v případě, že cílová strana vyžaduje pokaždé vyplnit některé klíče v těle požadavku, které by ale byly stejné napříč všemi událostmi, které se do služby zasílají, a tak by se v případě, že by se společný mapping nevyužil, musely klíče předvyplňovat na všech entitách mappingu zvlášť, což by akorát postup znepřehlednilo.

Sloupec `max_concurrency` udává maximální počet otevřených HTTP požadavků na tento cíl paralelně vedle sebe a sloupec `rate_limit_cooldown_ms` udává v milisekundách čas, který se má počkat v případě, že cíl odešle stavový kód HTTP 429, který znamená „Too Many Requests“, neboli příliš mnoho požadavků najednou. Trackman detekuje tento chybový kód a pozastaví odesílání všech HTTP požadavků na tento cíl do doby, než uběhne zde definovaný čas, a pak pokračuje v odesílání.

Poslední dva sloupce jsou spravované přímo databází a mají pouze informativní charakter. Jedná se o sloupce `created_at` a `updated_at`, které uchovávají informaci o čase vytvoření, respektive poslední úpravy daného cíle.

4.2.1.2 Event

Tabulka `trackman.event` obsahuje ve svých řádcích seznam událostí, které má Trackman očekávat ke zpracovávání. Obsah této tabulky ovlivňují ostatní microservices, které Trackmanovi při svém startu ohlašují, jaké události od nich může očekávat. Toto ohlášení probíhá automaticky na bázi konfiguračního souboru `trackman.json`, který vyplňuje vývojář dané služby. Při startu, při navazování spojení s databází dojde automaticky (díky podpoře Trackmana implemento-



Obrázek 4.3 Tabulka `trackman.event`

vané na úrovni databázového ovladače `@integromat/pg`) k zavolání synchronizační procedury `public.trackman_events_sync` ve veřejném databázovém schématu, která následně zařídí sesynchronizování s tabulkou `trackman.event`. Trackman umí odesílat i události takzvaně „anonymní“, tedy ty, pro které nedošlo k jejich zaregistrování do této tabulky. Pro tyto události však není k dispozici nápověda a validace a je lepší tento způsob odesílání nevyužívat.

V tabulce najdeme sloupec `name`, který je zároveň primárním klíčem této tabulky. Vzhledem k tomu, že řádky do tabulky synchronizuje dříve zmíněná procedura, není zde potřeba mít číselný primární klíč, který by byl poněkud zbytečný. Název události musí totiž být unikátní v rámci celé platformy a tak ho můžeme v této tabulce používat právě jako primární klíč. V případě kolize vypisuje synchronizační procedura varování a vývojář se tak dozví, že názvy událostí, které používá, už jsou zabrané a měl by tedy své přejmenovat. Dále obsahuje sloupce `label` a `description` které slouží pro snadnější orientaci v poskytovaných událostech.

Sloupec `payload` pak obsahuje popis datové struktury události. Tento popis se následně používá i k validaci příchozích událostí před tím, než je na nich provedeno mapování. Posledním sloupcem je sloupec `provider`, který nese jméno mikroslužby, která danou událost zaregistrovala. Rovněž tak slouží pro snadnější orientaci v procesovaných událostech.

4.2.1.3 Event Target Mapping

Tabulka `trackman.event_target_mapping` je relační tabulka spojující tabulky `trackman.event` a `trackman.target`. Dekomponuje *M:N* vazbu mezi entitami *Event* a *Target*. Jelikož Trackman podporuje také takzvané „anonymní události“, tak zatímco na tabulku `trackman.target` je zde reference silná, zařízená přes referenční integritu a cizí klíč, v případě tabulky `trackman.event` je zde reference pouze formální, tedy aplikační vrstva se při vyhodnocování mapování snaží danou událost nalézt, je však počítáno s tím, že nalezena být nemusí. Dále tato tabulka přidává na zmiňovanou relaci další vlastnosti, kterými jsou:

- `mapping`, což je samotná definice mapování dané události na daný cíl a je zapsána pomocí IML syntaxe.
- `condition`, rovněž v zápisu IML, je výraz, který se před odesláním události musí vyhodnotit na *truthy* nebo *falsy* hodnotu a na základě toho je buď odeslání provedeno, či nikoli.

- `dump`, typu `boolean`, je vlajka, která specifikuje, zdali má být při zpracování tohoto mapování do konzole vypsán podrobný výstup prováděných operací. V produkčních prostředích je zpravidla vypnutý, ale velice se hodí při ladění a testování nových událostí.
- `enabled`, rovněž typu `boolean`, je vlajka říkající, zdali je dané mapování aktivní, nebo ne.
- `created_at` a `updated_at` jsou pouze kontrolní sloupce a udržují informaci o čase vytvoření, respektive poslední úpravy řádku.

4.2.1.4 Queue

Tabulka `trackman.queue` je nejvytíženější tabulkou v rámci celého řešení. Skrze tuto tabulku totiž protékají všechny události, které Trackman zpracovává. Dalo by se namítnout, že místo této tabulky mohla být zvolena nějaká frontovací služba, která by si s velkou zátěží jistě poradila lépe, než to zvládne databáze PostgreSQL, i když ani ta si ve velké produkční zátěži nevede vůbec špatně.

Nutno říci, že v době, kdy vznikala specifikace pro službu Trackman, počítalo se ještě minimálně s jedním až dvěma lety běhu v rámci platformy Integromat 1.0, která vzhledem ke svému technologickému řešení nenabízela možnost nasazení fronty, jako mohla být například **RabbitMQ**. Proto bylo rozhodnuto, že bude vytvořena tato frontovací tabulka a Trackman má tak, výměnou za to, závislost pouze na databázi a žádných dalších externích službách.

Definice sloupců tabulky `trackman.queue`:

- `id`, primární klíč tabulky, automaticky generovaná číselná sekvence
- `event_name`, název události ke zpracování
- `payload`, tělo události, vlastní zaznamenaná data k odeslání
- `skipped_until`, časová značka říkající *do kdy* se má událost ve frontě přeskakovat
- `fails_count`, udává počet odeslání, které selhaly, používá se při počítání v retenčním mechanismu
- `failed_targets`, obsahuje identifikátory cílů, na něž odesílání selhalo
- `priority`, udává prioritu zprávy ve frontě
- `original_id`, využívá se při znovu-zařazení chybné události zpět do fronty, obsahuje hodnotu sloupce `id` z původní události
- `created_at` je systémový sloupec a obsahuje časové razítko vložení události do fronty

4.2.2 Procedury

Pro manipulaci s daty v tabulkách používá Trackman **výlučně** uložené databázové procedury. Jsou napsány v jazyce `plPGsql` a tvoří komunikační rozhraní mezi aplikačním JavaScriptovým kódem a daty, která jsou v tabulkách uložena. Mnoho logiky je ve skutečnosti uloženo až na vrstvě procedur, nikoliv v aplikačním kódu jako takovém, a tento přístup velice zefektivňuje a zjednodušuje práci s datovým modelem a obě strany kódu, jak JavaScript, tak SQL, jsou díky tomuto přístupu velice přehledné a snadno testovatelné.

Správa procedur je v Trackmanovi řešena skrze interní ovladač `@integromat/pg`, který kromě jiného obsahuje také komponentu **Evoman**, což je akronym pro Evolution Manager, neboli správce databázové evoluce. Tato komponenta zajišťuje při každém startu aplikace spojení s databází a synchronizaci databázových procedur, které Evomanu při spuštění aplikace předá.

Dovolil bych si zde porovnat přístup uložených procedur s přístupem ORM a také vysvětlit, proč byl zvolen právě přístup uložených procedur a jaké z něj plynou v tomto ohledu výhody. Objektově relační mapování je nepochybně úžasný nástroj, pokud aplikace pracuje s velkým množstvím entit najednou a s mnoha různými stavy, které entity mohou nabývat. Pokud si vezmeme příklad z jazyka Java a knihovny Hibernate, jedná se o skvělý ORM framework, který vývojáři opravdu usnadní práci s databází. Základní myšlenkou při ORM tedy je načtení entity z databáze, práce s entitou na aplikační vrstvě, a její následné uložení zpět bez toho, aby programátor musel ovládat jazyk SQL. Vše, ať už DDL příkazy, nebo vlastní DML příkazy, se tedy generuje na pozadí.

Důvod, proč je v našem případě využití uložených procedur daleko efektivnějším přístupem je, že v databázi entity pouze neukládáme, ale rovnou s nimi pracujeme na databázové vrstvě. Díky tomu, že jsme schopni přenést značnou část, řekl bych až možná polovinu logiky, na úroveň databáze, celý proces se rapidně zrychluje, protože například pro změnu stavu entity, i netriviální, stačí pouze zavolat odpovídající databázovou proceduru a předat jí parametry, ale nedochází k datovému přenosu celé entity mezi databázovou a aplikační vrstvou. Databáze zároveň díky transakčnosti sama pohlídá konzistenci dat. Dalším argumentem je fakt, že z univerzality ORM naopak vyplývá i jeho nevýhoda. Pokud daný ORM framework podporuje například čtyři různé databázové stroje, může pohodlně poskytovat funkcionalitu jen ze společného průniku funkcionalit těchto dílčích strojů. Přístup, kdy si databázové procedury píšeme sami a na míru jednomu konkrétnímu databázovému stroji, nám dává velmi silnou kontrolu nad tím, co přesně se na úrovni databáze stane a umožňuje nám vytěžit z výkonu a funkcí databázového stroje maximum.

Aplikace obsahuje celkem přes padesát různých procedur, které pomáhají udržet konzistentní datový model a zajišťují správu vlastní dat. Několik klíčových bych zde rád představil.

4.2.2.1 trackman.queue_next

Procedura `trackman.queue_next(_max_events integer)` je základním kamenem celé trackovací smyčky, která bude rozebrána později. Tato procedura zajišťuje vyzvednutí událostí k odbavení z fronty událostí. S využitím konstrukcí `SELECT FOR UPDATE` a `SKIP LOCKED` je napsána takovým způsobem, že vrácené řádky do konce transakce zamyká a zároveň přeskakuje již zamčené. Toto je velice podstatný koncept. Služba Trackman je od začátku zamýšlena pro clusterový běh ve vícero instancích. Pokud by k zamykání nedocházelo, mohlo by se stát, že se dvě instance zeptají fronty na nové události ve stejnou chvíli a obě instance pak dostanou sady událostí s neprázdným průnikem, což by efektivně vedlo k duplikaci odeslaných dat, což samozřejmě z různých důvodů nechceme. Jelikož PostgreSQL přiděluje zámky na tabulce sériově tak, jak dotazy přicházejí, vždy se jedna z instancí musí zeptat dříve než druhá, čehož právě tato procedura využívá. Instance, která proceduru zavolá, pomocí konstruktu `FOR UPDATE` zároveň vrácené řádky zamkne. Druhá instance v pořadí už ale naopak díky využití `SKIP LOCKED` tyto řádky nevidí a dostane tedy další set. Tímto způsobem elegantně a bez kolizí může vícero instancí volat tuto proceduru paralelně. V případě, že dojde k aplikační chybě, dochází k rollbacku celé transakce, řádky se odemykají a dostane je ke zpracování další instance. V případě, že se odeslání povede, dochází k operaci `COMMIT` a řádky jsou z databáze smazány.

4.2.2.2 trackman.events_sync

Procedura `trackman.events_sync(_provider character varying, _events jsonb)` se stará o synchronizaci událostí poskytovaných ostatními službami v aplikačním clusteru. Opět s využitím vlastního databázového ovladače. Když jakákoliv z komponent startuje, testuje se databázová konfigurace na existenci souboru `trackman.json`. Tento soubor obsahuje definice událostí, které má Trackman od dané služby očekávat. Databázový ovladač pak autonomně vyhledá v databázi schéma `trackman` a proceduru `events_sync`. Pokud tyto existují, zavolá proceduru a předá jí seznam událostí poskytnutých třetí stranou. `events_sync` provede zasynchronizování s tabulkou `trackman.event` a vrátí výsledek, zda se povedlo všechny události sesynchronizovat, případně

```
CREATE OR REPLACE FUNCTION trackman.queue_next(_max_events integer DEFAULT 512,
                                               OUT id bigint,
                                               OUT payload jsonb,
                                               OUT priority smallint,
                                               OUT event_name character varying,
                                               OUT skipped_until timestamptz,
                                               OUT fails_count smallint,
                                               OUT failed_targets integer[],
                                               OUT original_id bigint,
                                               OUT created_at timestamptz)
        RETURNS setof record
        LANGUAGE plpgsql
        VOLATILE
        SECURITY DEFINER AS
$$
BEGIN

    RETURN QUERY
        DELETE
            FROM trackman.queue q
                WHERE q.id IN (
                                SELECT r.id
                                FROM trackman.queue r
                                WHERE r.skipped_until IS NULL
                                   OR r.skipped_until < CURRENT_TIMESTAMP
                                ORDER BY r.priority DESC, r.created_at
                                LIMIT _max_events
                                FOR UPDATE SKIP LOCKED
                            )
                RETURNING q.id,
                          q.payload,
                          q.priority,
                          q.event_name,
                          q.skipped_until,
                          q.fails_count,
                          q.failed_targets,
                          q.original_id,
                          q.created_at;

END;
$$;
```

Výpis kódu 4.1 Ukázka procedury `trackman.queue_next`

```
CREATE OR REPLACE FUNCTION public.trace(_event_name character varying,
                                        _payload jsonb,
                                        _priority smallint DEFAULT 10)
 RETURNS void
 LANGUAGE plpgsql
 SECURITY DEFINER
AS $$
BEGIN

    IF _priority IS NULL THEN
        _priority = 10;
    END IF;

    PERFORM trackman.queue_push(_event_name, _payload, _priority);
EXCEPTION
    WHEN OTHERS THEN RETURN;
END;
$$;
```

Výpis kódu 4.2 Ukázka procedury `public.trace`

zda někde došlo k chybě či kolizi. Tato funkcionalita velice zpříjemňuje práci vývojářům ostatních komponent, kteří nemusí řešit implementační detaily předávání událostí na straně Trackmana, ale pouze spravují `trackman.json` soubor ve své komponentě a přes něj si pak Trackman zbytek informací zjistí sám.

4.2.2.3 public.trace

Klíčovou procedurou, bez které by vůbec nefungovalo sledování událostí, jak pak procedura `public.trace(_event_name character varying, _payload jsonb)`. Tato procedura je umístěna ve veřejném databázovém schématu, tedy k ní má přístup úplně každý uživatel Integromat databáze, respektive každá běžící služba, která je k databázi připojena. Procedura přijímá název události a vlastní datový obsah. Je napsána tak, aby sama ošetřila případnou neexistenci `trackman` schématu, jelikož se může stát, že bude daná komponenta spuštěna v prostředí, kde Trackman neběží a tedy by nebylo odpovídající databázové schéma vytvořeno. Ve finále jde o velmi jednoduchý kód, který přijatá data zafrontuje do tabulky `trackman.queue`, odkud si je pak v rámci sledovací smyčky vyzvedne dříve zmíněná procedura `trackman.queue_next` a dojde k jejich distribuci.

4.3 Aplikační vrstva

Aplikační vrstva služby Trackman je naprogramována v jazyce JavaScript, primárně s využitím serverové implementace NodeJS. Zde se využívá knihovna `@integromat/express`, která je jistou abstrakcí a modifikací originálního JavaScriptového frameworku `express`, který je v současnosti patrně nejpoužívanějším frameworkem pro stavění HTTP REST API aplikací v jazyce JavaScript.

Velkým přínosem této knihovny je například integrovaná podpora clusterových běhů skrze NodeJS rozšíření `child-process`, což dává Trackmanovi možnost velice jednoduše přepnout z jednoinstančního běhu do běhu ve více instancích, které jsou všechny spravovány takzvaným `manager` procesem. Jak již bylo zmíněno, Trackman je plně připraven na běh ve více instancích paralelně a díky důmyslnému využití vlastností zámků v PostgreSQL se aktivně brání kolizím při čtení a zápisu dat.

Nad knihovnou Integromat Express je následně stavěna vlastní aplikační logika, tu bych zde nyní rád podrobněji rozebral.

4.3.1 Obecná architektura aplikace

Aplikace je rozdělena do několika logických celků. Dalo by se dokonce uvažovat vícero možných členění. Úplně základním rozdělením backendové architektury v případě Trackmana je část kontrolní a pak část procesní.

4.3.1.1 Kontrolní část

Kontrolní část slouží pro ovládání aplikace Trackman. Vzhledem k tomu, že nemalá část logiky je uložena přímo v databázových procedurách, dalo by se říci, že Trackman má dvojí ovládací API.

Nižší ovládací vrstvou je API databázové, které je tvořeno databázovými procedurami, které obsahují klíčové kusy logiky nezbytné pro operace s daty. Na úrovni těchto procedur je vyřešeno ukládání dat, jejich zpracování, načítání, hlídání kolizí a mnoho dalšího. Trackman může být tedy, až na pár výjimek, ovládán čistě voláním databázových procedur.

Druhou, vyšší ovládací vrstvou Trackmana je HTTP REST API. Toto API přidává nad původní databázové API například pokročilejší validaci. Zatímco u ovládání přes databázové API dojde v případě zaslání neplatného požadavku k zachycení výjimky až na úrovni databázové operace, API validace vůbec nedovolí, aby byl chybný požadavek do databáze odeslán. Řeší třeba povinné parametry jednotlivých volání, ale zvládne i pokročilejší validace, například na základě regulárních výrazů, které musí parametr splňovat. K popisu validního formátu je využíván zápis `Integromat Data Structure Syntax` a následnou validaci řeší interní knihovna `@integromat/blueprint`, respektive její komponenta `TypeValidator`.

Další výhodou HTTP API oproti databázovému je, že umí pracovat se šifrovanými daty. Některá data se v databázi ukládají šifrovaně. Jde především o přístupové API klíče k účtům třetích stran, kam se následně odesílají zpracované události. V databázi jsou tato data uložena v binární podobě, v datovém typu `bytea`, a databáze jako taková nemá možnost data přečíst, respektive rozšifrovat. O šifrování dat se stará aplikační vrstva, konkrétněji knihovna `@integromat/pg`, která jako jednu z funkcionalit podporuje právě šifrování a dešifrování přenášeného obsahu. Jelikož k šifrování dochází už na aplikační vrstvě, HTTP API může, na rozdíl od DB API, zpracovávat i šifrovaná data, a je tedy vhodné právě pro nastavování přístupových klíčů a hesel.

Kontrolní část je tedy tvořena `routerem`, který je specifikovaný v souboru `api.js` a následně k němu přidruženými kontrolery, které obsahují handlery pro jednotlivé API požadavky. Kontrolerů je v aplikaci několik a jsou rozděleny podle toho, za jakou část zodpovídají. Najdeme tedy například kontroler pro správu cílů, správu mapování a mnoho dalších.

4.3.1.2 Autorizace

Ke kontrolní části patří také autorizace. Trackman podporuje dva různé způsoby autorizace, a to Session Autorizaci a autorizaci pomocí API klíčů.

Session Autorizace úzce souvisí s prostředím, kde služba Trackman reálně běží. Počítá se totiž s tím, že Trackman běží jakou součást aplikačního clusteru platformy Integromat, tedy běží ve stejném prostředí jako třeba webová aplikace nebo centrální API Integromatu. Když je uživatel přihlášen do Integromatu jako do aplikace, Trackman se umí napojit na Redis databázi, kam se uživatelské sessions ukládají, a umí je následně využít i pro autorizaci ke svému kontrolnímu API.

Pokud se tedy do aplikace Integromat přihlásí administrátor, Trackman sám umí při pokusu o přístup z tohoto účtu zkontrolovat, jaká práva daný uživatel má a pokud mezi jeho práva patří

oprávnění ke správě konfigurace Trackmana, je jeho požadavek propuštěn dále. Naopak, pokud uživatel přihlášen není nebo nemá patřičná oprávnění, je API požadavek zakončen s chybovým stavem `HTTP 401`, respektive `HTTP 403`.

Druhou možností je autorizace přes API klíč. API klíč se dá vygenerovat skrze databázové API a počítá se s tím, že ho uživateli nebo službě, která jej vyžaduje, vyrobí administrátor systému. Uživatel nebo služba se pak následně místo session pro Trackmana autorizuje právě zasláním API klíče v autorizační hlavičce. Na úrovni API klíčů není implementována žádná pokročilá logika oprávnění, API klíč, stejně jako ostatně i Session Autorizace, dávají uživateli absolutní moc správy aplikace. Na vyšší granularitu oprávnění nebyl v současné době požadavek. Trackman tedy zkusí z příchozího API požadavku získat API klíč, pokud je klíč nalezen a je validní, je požadavek propuštěn dále, v opačném případě opět dojde k uzavření se stavovým kódem `401`.

Díky návrhovému vzoru Middleware, který používá využitý framework Express, je provedení autorizace uživatele pak z pohledu celé aplikace velice jednoduché. Před napojením samotného API routeru je zapojena dvojice middlewarů `detectAuthorization` a `authorize`, které v sobě zapouzdřují veškerou autorizační logiku.

4.3.1.3 Procesní část

Procesní část je klíčovou součástí celé aplikace Trackman a obsahuje hlavní výkonnou logiku, která je vlastním jádrem celé služby a tedy potažmo i této práce, proto budu jednotlivým komponentám procesní části věnovat následujících několik podkapitol.

4.3.2 Architektura smyček

Základní myšlenkou celé procesní části backendového řešení Trackmana jsou takzvané smyčky. Než se pustím do jejich popisu, jenom pro úplnost zmíním, že doba „tikání“ jednotlivých smyček je plně parametrizovaná do konfigurace služby a dá se tak měnit za běhu bez nutnosti změny v kódu.

Trackman tedy stojí na smyčkách a každá z nich vykonává nějakou z klíčových rutin, které je potřeba obsluhovat.

- **Monitorovací smyčka**
  Jde pouze o podpůrnou smyčku a proto ji zde zmíním jenom krátce. Při každém běhu zjistí stav obou front[3] a rovněž spočítá stav cache a všechny tyto agregované statistiky odešle na monitoring a na daný časový interval je zacachuje. Díky této smyčce pak v externím monitoringu vidíme délky front v průběhu času a další interní statistiky.

- **Mapovací a Webhooková smyčka**
  Obě tyto smyčky zajišťují obnovení odpovídajícího mapovacího, respektive webhookového stromu v paměti a velice tak pomáhají s optimalizací výkonu. Stromům, které Trackman v paměti udržuje, bude věnována následující podkapitola. Zde je důležité zmínit, že obnovování stromů, kromě jiného, zajišťují také tyto smyčky.

- **Trackovací smyčka**
  Zodpovídá za vlastní odbavení aktuálního úkolu, respektive aktuálního balíku událostí z odpovídající fronty. Každé tiknutí této smyčky vede k odeslání dat na připojené třetí strany. Smyčka si zároveň hlídá, aby se nespustila znovu, pokud ještě nedoběhlo předchozí vyhodnocení. Zvláště u zpracovávání dat z CRONů může velice jednoduše nastat situace, kdy samotné provedení úkolu trvá déle, než jak dlouhý je interval mezi dvěma tiky smyčky. Pokud smyčka vyhodnotí, že se tedy může spustit, předá řízení funkci `track`, což je hlavní vstupní bod celé trackovací logiky. Detailněji trackovací smyčku rozeberu v podkapitole Trackovací smyčka podrobně.

[3] Tedy jak fronty úkolů, tak fronty událostí

4.3.3 Stromy v paměti

V rámci paměťové a rychlostní optimalizace si Trackman udržuje v paměti dva „stromy“. Jsou jimi Mapovací Strom a Strom Webhooků. Tyto stromy jsou obnovovány zpravidla periodicky, pomocí smyček, které byly zmíněny již dříve, ale také dochází k jejich automatickému obnovení ve chvíli, kdy dojde ke změně podstatných dat v databázi. Logika databázových procedur umí při uložení nových dat odeslat skrze frontu úkolů zprávu aplikační vrstvě, která na základě této zprávy provede obnovení daného stromu.

Optimalizace spočívá v následujících krocích. Stejně jako na několika jiných místech v rámci aplikace je mapování událostí na cíle provedeno s využitím syntaxe IML a stejnojmenné knihovny. Proces vyhodnocení IML výrazu se skládá ze dvou kroků - `parse` a `execute`.

Krok `parse` přijímá řetězec nebo objekt, který je zapsán v jazyce IML. Tento následně zpracuje, naparsuje, a vytvoří abstraktní syntaktický strom – AST. Funkce IML `execute` následně přijímá dva klíčové parametry – AST a kontext. Kontextem se myslí slovník hodnot, které mohou být v rámci objektu mapovány a AST je přesně zmíněný výsledek z kroku `parse`.

Zatímco pro každou událost logicky může výsledek IML transformace vypadat úplně jinak, v závislosti na datech, která událost obsahuje, datová struktura výsledku je vždy stejná - nezměnilo se totiž mapování, ale pouze vstupní data. Z pohledu funkce Execute se tedy mění pouze Kontext, ale AST zůstává neměnný pro všechny události daného typu.

Zde se právě procesu účastní mapovací strom. Při stavbě mapovacího stromu dojde k zavolání databázové procedury, která aplikační vrstvě vrátí seznam všech dostupných mappingů, které jsou v databázi uložené. Aplikace následně pro každý jednotlivý mapping provede `IML.parse`, čímž mapping dostane do podoby AST a je tak připraven na provádění transformací. Výsledek, tedy AST, je následně uložen do paměti a mapovací strom je pomocí reference poskytnut do funkce `track`, která jej následně využije k vlastnímu mapování. Pro každou událost už ovšem vždy dojde pouze k zavolání `execute` s daným kontextem události, ale tímto předparsováním bylo ušetřeno mnoho volání funkce `parse`, která je místo toho volána pouze jednou, když dochází ke stavbě stromu v paměti.

Druhotná optimalizace zde spočívá v tom, že nedochází k opakovanému zjišťování stavu mapování z databáze, ale rovněž pouze z paměti. I tady se může tato optimalizace pozitivně projevit na rychlosti celého procesu, jelikož jak volání `IML.parse`, tak volání databáze, nejsou triviální funkce a jejich běh může zabrat i stovky milisekund. Kdyby se toto volání mělo dít pro každou jednu událost, kterých Trackman během jednoho dne vyhodnocuje stovky tisíc, došlo by k výraznému zbrzdění celého zpracování.

U stromu Webhooků je situace podobná. Stejně jako mapování, i instrukce pro zpracování příchozích webhooků jsou uloženy v databázi a opět jsou zapsány jazykem IML. Princip optimalizace je stejný. K postavení stromu webhooků dochází v paměti jednou za čas a nebo při detekované změně a následně se celé namapování příchozí zprávy provádí už pouze z paměti a nevyvíjí tak žádnou zátěž na databázi.

Důležitou výhodou smyček, kterou bych zde rád zmínil, je fakt, že běží asynchronně. To znamená, že stavba nového mapovacího stromu neblokuje například trackovací smyčku. K asynchronnímu spuštění smyčky dochází přes rozšíření knihovny Integromat Express, kde je zadán takzvaný `repeatableTask`, respektive `silentRepeatableTask`. Tyto úlohy interně využívají NodeJS rozhraní `setInterval`.

4.3.4 Trackovací smyčka podrobně

Trackovací smyčka je nejdůležitější komponentou celé aplikace Trackman. Jedná se o místo, kde dochází k vyhodnocování jednotlivých přijatých událostí, k jejich transformaci a k následné distribuci na cíle, tedy na analytické nástroje třetích stran.

Vše začíná voláním databázové procedury `trackman.queue_next`, která jako jediný vstupní parametr přijímá počet událostí, které má zamknout a připravit k odeslání. Výstupem z této

procedury je pro aplikační vrstvu pole událostí, které se mají zpracovat. Následně dochází ke spuštění cyklu pro každou tuto událost.

Prvním krokem je zjištění, zda pro danou událost vůbec existuje v mapovacím stromě nějaké mapování, které by bylo použitelné. Pokud není žádné mapování nalezeno, cyklus automaticky pokračuje na další událost, protože s tou aktuální již víc udělat nejde.

Pokud je nalezeno alespoň jedno mapování, dochází k validaci těla události. V případě, že tělo události neodpovídá validačnímu schématu, dojde k ohlášení výjimky a pokračuje se další událostí. Nutno podotknout, že v případě potřeby se dá validační krok v konfiguraci služby Trackman vypnout, což může být vhodné pro testování a ladění, kdy je potřeba propouštět i neplatné záznamy.

Nyní jsme tedy ve stavu, kdy máme k dispozici alespoň jedno mapování a víme, že tělo události je validní. Mapování může být k dispozici vícero, jelikož bez problému můžeme chtít odesílat jednu událost na vícero různých cílů. Pro každé mapování tedy spouštíme další vnořenou smyčku.

Uvnitř tohoto vnořeného cyklu dojde k sestavení kontextu a k následnému provedení dříve zmíněného `IML.execute` nad připraveným AST z mapovacího stromu.

Následuje nejdůležitější fáze, a to samotné odeslání události. Jak bylo již na několika místech uvedeno, Trackman pro odeslání události na třetí stranu využívá moduly přímo z platformy Integromat, čímž velice pěkným způsobem recykluje již vytvořený kód a hlavně automaticky nabízí odesílání událostí na téměř všechny cíle, které samotná platforma Integromat podporuje k integraci.

Moduly aplikace Integromat jsou interně JavaScriptové třídy, které poskytují předem předepsané rozhraní. Trackman tedy postupně modul inicializuje, provede a následně finalizuje jeho běh. Za pouhým zavoláním `module.write` z pohledu kódu služby Trackman je schovaná veškerá logika sestavení API požadavku, jeho odeslání a přijetí a vyhodnocení odpovědi na něj. Díky této funkcionalitě dovede Trackman velice jednoduše odeslat data prakticky kamkoliv, kam by je šlo odeslat samotným Integromatem a nutno říci, že to je jedna z klíčových předností této služby.

4.3.4.1 Retenční mechanismus

Pokud vše proběhlo v pořádku, je událost označená jako úspěšně odeslaná a do fronty se již žádným způsobem nevrací. Pokud ovšem v jakékoliv fázi odesílání dojde k chybě, Trackman na chybu reaguje tak, že kromě ohlášení výjimky zároveň událost znovu zafrontuje k odeslání.

Pokud došlo k prvnímu selhání, je nový pokus o odeslání odložen o hodinu. Pokud došlo k selhání i podruhé, je třetí odeslání naplánováno o osm hodin později. Posledním krokem je znovuodeslání za čtyřiadvacet hodin. Pokud ani čtvrtý pokus o odeslání neprojde, dojde k zalogování velmi závažné chyby a událost je následně z fronty definitivně vyřazena. Nemožnost odeslání ani na čtvrtý pokus totiž indikuje chybu buď v mapování, nebo v obecné konfiguraci třetí strany a je nepravděpodobné, že by odeslání neprošlo z důvodu, že by protistrana byla čtyřikrát po sobě v různých časových intervalech nedostupná.

Kapitola 5
Testování a nasazení

Po úspěšné implementaci majoritní části služby Trackman přišla na řadu finální fáze, tedy její otestování a následné nasazení jak na vývojové, tak na produkční prostředí.

5.1 Jednotkové testy

Na klíčové komponenty Trackmana jsou vytvořeny jednotkové testy. Ty se spouští automaticky při každém commitu do Git repozitáře, kde je verzován samotný zdrojový kód. Pokud testy procházejí, dochází následně automaticky k sestavení Docker Image pro Git větev, do které ke commitu došlo.

K jednotkovým testům se na Trackmanovi využívá framework **Mocha** a nástroj **Chai**, což jsou jedny z nejpoužívanějších knihoven pro tento účel pro jazyk JavaScript.

Po úspěšném provedení jednotkových testů se rovněž spustí test integrační, který nechá aplikaci v testovacím prostředí kompletně naběhnout a následně ji opět ukončí. Tímto testem se otestuje několik klíčových aspektů, které by jednotkové testy testovaly pouze velmi složitě.

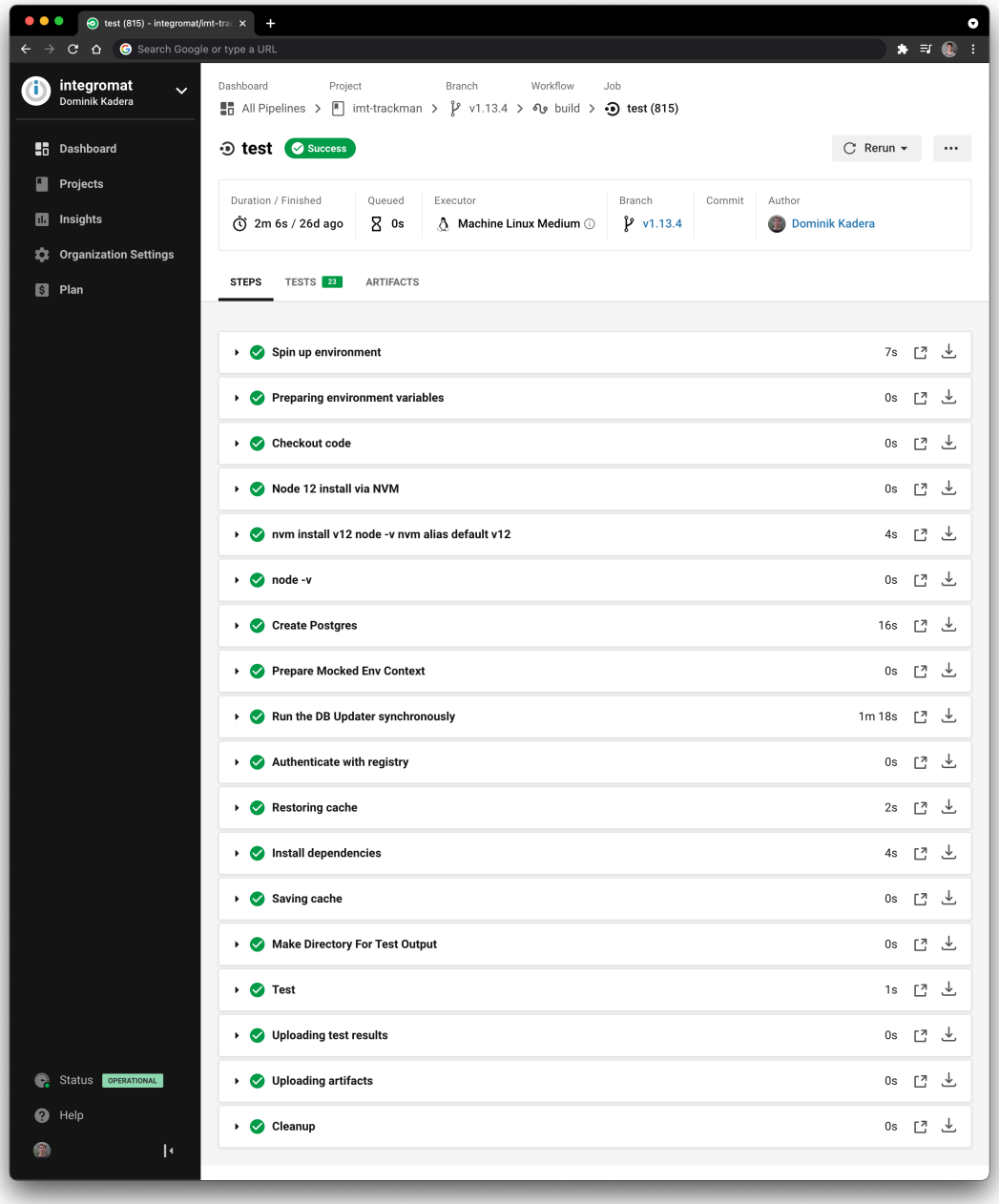
První důležitou věcí je testování správné synchronizace s databází. Interní funkcionalita databázového ovladače `@integromat/pg` při navazování spojení s databází totiž zároveň kontroluje a synchronizuje databázové procedury a evoluci databáze, podobně jako to například v Javě dělá framework Hibernate. Pokud by byl s databázovou evolucí problém, nedošlo by ke správnému spuštění aplikace a chyba by byla detekována. Už několikrát byl tímto způsobem nalezen problém přímo v databázovém ovladači, takže tento test zároveň nepřímo pomohl vylepšit i celkovou funkcionalitu platformy.

Druhou částí integračního testu, která může odhalit možné problémy, je test korektního ukončení aplikace. Mohlo by se totiž například stát, že aplikace po sobě správně neuzavře spojení s PostgreSQL či Redis databází, a zůstane tak na tomto spojení viset, dokud není po několika vteřinách násilně ukončena. Toto chování však jistě není žádoucí, a integrační test tento možný problém včas odhalí.

5.2 Testovací prostředí

Testovací instance Trackmana je následně spuštěna na vývojovém prostředí Integromatu. Toto prostředí obsahuje vesměs všechny běžící komponenty, ovšem běžící z vývojových větví a se zapnutými pokročilejšími možnostmi logování.

Na tomto prostředí v rámci celého procesu vývoje přicházejí s Trackmanem do styku kolegové z *Growth Teamu*. Na analytických nástrojích, kam Trackman data odesílá, jsou rovněž vytvořeny testovací pracovní prostory, kam je služba připojena. *Growth Team* tedy následně vytváří nové



Obrázek 5.1 Testovací Pipeline v CircleCI

specifikace událostí, které je potřeba pro podporu růstu v rámci platformy sledovat. Události jsou vývojářem implementovány do kódu služby, která bude zodpovědná za jejich zaznamenávání. Tyto události začnou automaticky chodit Trackmanovi, ve kterém se nakonfiguruje správné mapování a data tak dorazí až na třetí stranu.

Na vývojovém prostředí tedy probíhá nejen testování služby Trackman v reálném provozu uvnitř clusteru, ale také se zde testují její konfigurační možnosti a události, které skrze službu protékají.

5.3 Reálné nasazení

V době tvorby této práce je již služba Trackman nasazena i na produkční prostředí Integromatu 1.0. Funguje zde jako klíčová komponenta pro sledování akcí v platformě a na základě jeho výsledků již bylo vytvořeno mnoho nápověd a dalších prostředků, které podporují uživatelský zážitek a snaží se uživateli cestu Integromatem co nejvíce zpříjemnit.

Trackman na produkčním prostředí:

- Odesílá data na celkem **4 různé** analytické nástroje, do každého z nich proudí jiná množina událostí s vlastním mapováním tak, aby výsledek co nejlépe splňoval požadavky a očekávání týmu pro podporu růstu
- Přijme za **1 týden** zhruba **1.5 milionu** událostí ke zpracování
- Vyšle za **1 týden** zhruba **3.9 milionu** API požadavků na třetí strany
- Má ve třicetidenním průměru **99.98%** úspěšnost odeslání události na první pokus

Všechny tyto statistiky mohly být získány díky tomu, že Trackman využívá knihovnu Overseer a reportuje během svého běhu velmi podrobné statistiky o tom, co se zrovna provádí a zpracovává.

Závěr

V době, kdy píši tuto část práce, už fungující verze služby Trackman běží nasazena v produkčním prostředí Integromatu a produkuje reálné výsledky. Nad daty odesílanými touto službou probíhají v několika službách třetích stran analýzy chování, na základě kterých zobrazujeme našim uživatelům kvalitnější obsah, ať už se jedná o přizpůsobenou nápovědu či průvodce platformou.

Díky možnosti využívat konektory aplikace Integromat přímo v Trackmanovi bylo dosaženo maximální možné míry abstrakce, kdy je služba schopná využívat všechny dostupné aplikace, které Integromat nabízí, a díky tomu je přidání nového cíle od myšlenky až k realizaci otázkou několika hodin, což je v produkčním prostředí skutečně potěšující.

S využitím veřejného schématu v aplikační Postgres databázi je velice snadné pro jakoukoliv jinou komponentu v rámci Integromatu začít posílat skrze Trackmana nový typ události a díky velice vysoké modularitě celého kódu není pro přidání nového eventu potřeba zásah do codebase Trackmana jako takového.

Ovládání má dvě ovládací úrovně. Jednu low-levelovou, za pomoci databázových procedur, která je dostupná pouze administrátorům a druhou vyšší, pro správce Trackmana, přes webové HTTP REST API, které využívá sdruženou autorizaci se zbytkem platformy a nabízí také přihlašování pomocí API klíčů.

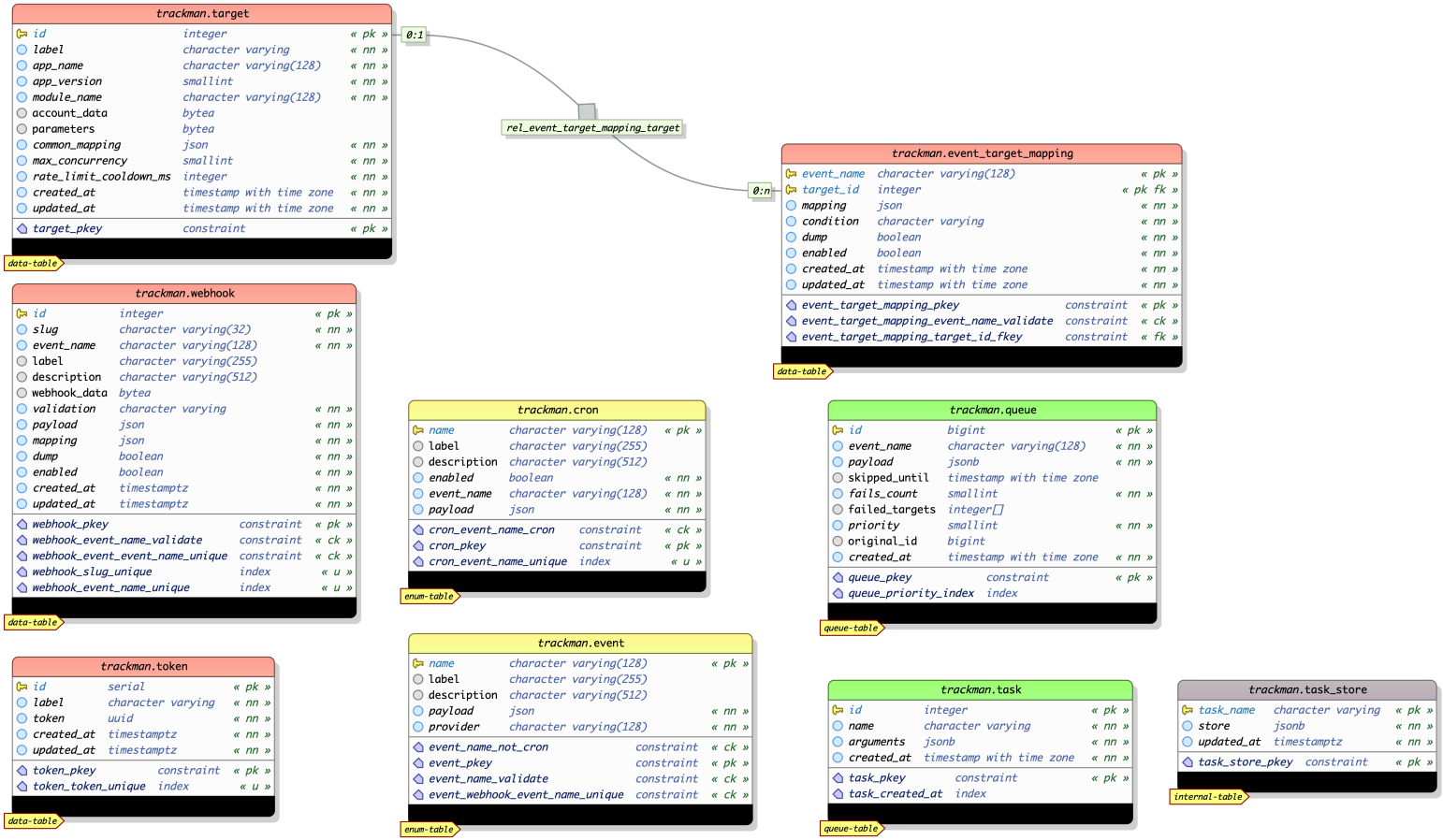
Obecně považuji všechny požadavky dané zadáním za naplněné a celkový cíl Trackmana nejen za dosažený, ale zároveň za překonaný. Díky všem možnostem, které mi zvolená architektura dala, jsem vytvořil službu, která teď zajišťuje v produkčním prostředí důležitou funkcionalitu a je neustále rozšiřována.

Příloha A

Databázový model

Celou databázi k Trackmanovi jsem navrhl s využitím nástroje PgModeler[1]. Na následujícím listě přikládám nákres databázového modelu, který jsem sestavil.

[1] https://pgmodeler.io/



Obrázek A.1 Model databázového schématu `trackman`

Příloha B

Aplikace třetích stran

Během vývoje i nasazení aplikace Trackman bylo využíváno služeb různých třetích stran, ať už se jednalo o analytické nástroje, nebo například o monitorovací služby. V této příloze bych na tyto služby rád odkázal, aby si čtenář práce mohl udělat obrázek, co která z těchto služeb vlastně znamená.

B.1 Customer.io

Jedná se o platformu pro správu uživatelské základny, která klade důraz na mailing a komunikaci s koncovými uživateli. Umožňuje uživatele rozdělovat do skupin a segmentů automaticky, na základě jejich chování. Informace o tom, jak se uživatelé chovají, jsou předávány Trackmanem skrze HTTP API této služby. Na základě příchozích událostí lze potom stavět různorodé emailové kampaně.[2]

B.2 Userflow

Tato služba se zaměřuje na optimalizaci onboarding procesu[1]. Pomocí eventů, které jí Trackman zasílá, je schopna vyhodnocovat chování uživatelů a následně ve webové aplikaci ukazovat různá doporučení a nápovědy, když dojde ke zjištění, že by se to uživateli potenciálně mohlo hodit.[11]

B.3 Datadog

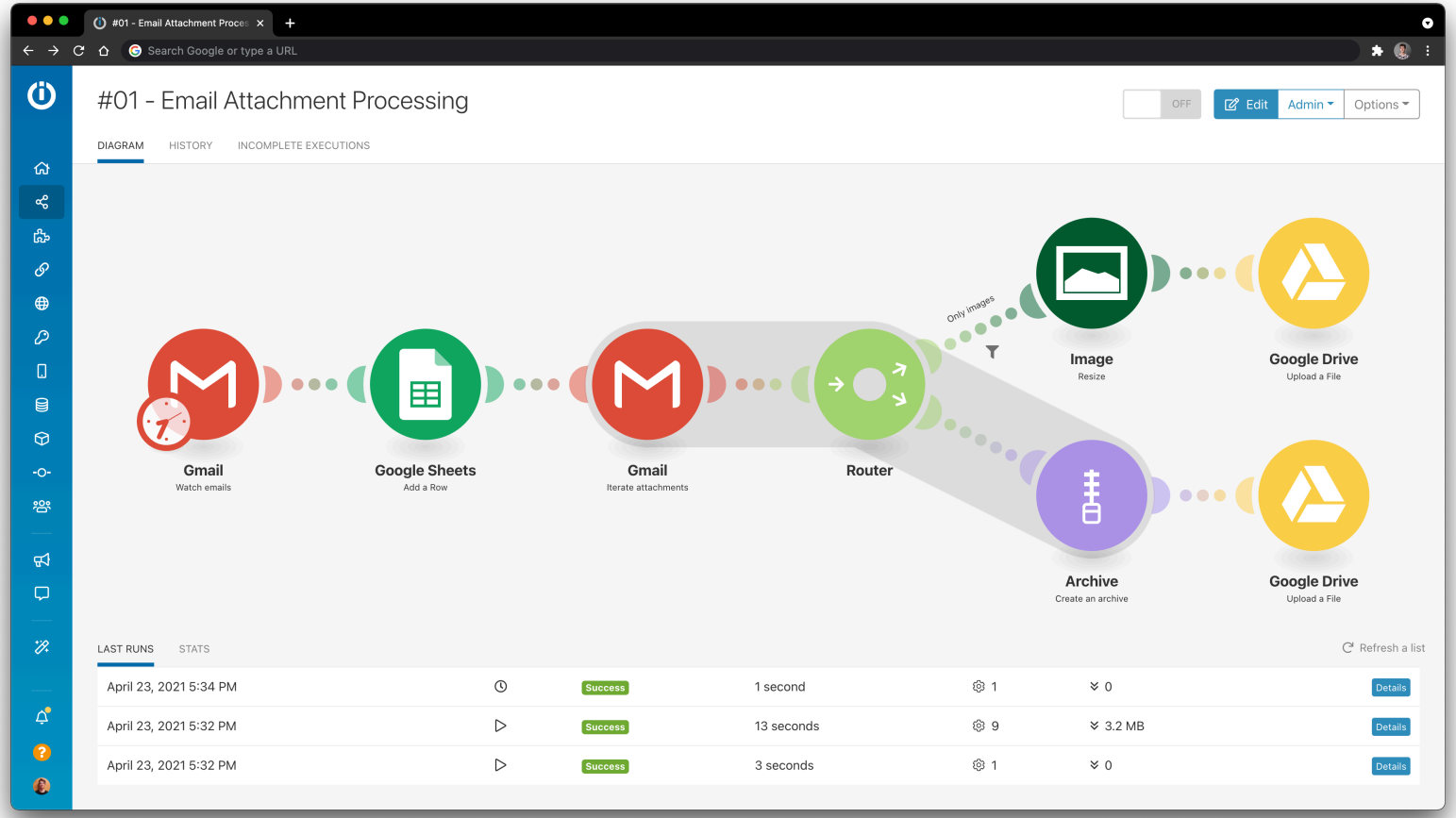
Datadog je moderní monitorovací platforma pro cloudové aplikace. Pomocí jednoduchého API umožňuje přijímat širokou paletu různých metrik poskytovaných monitorovanou aplikací, a tyto nadále zpracovávat a vyhodnocovat. Rovněž nabízí pokročilé možnosti monitoringu a alertingu, takže je ideální i pro automatickou detekci anomálií a incidentů.[3]

[1] Proces *nalodění* nového uživatele do produktu

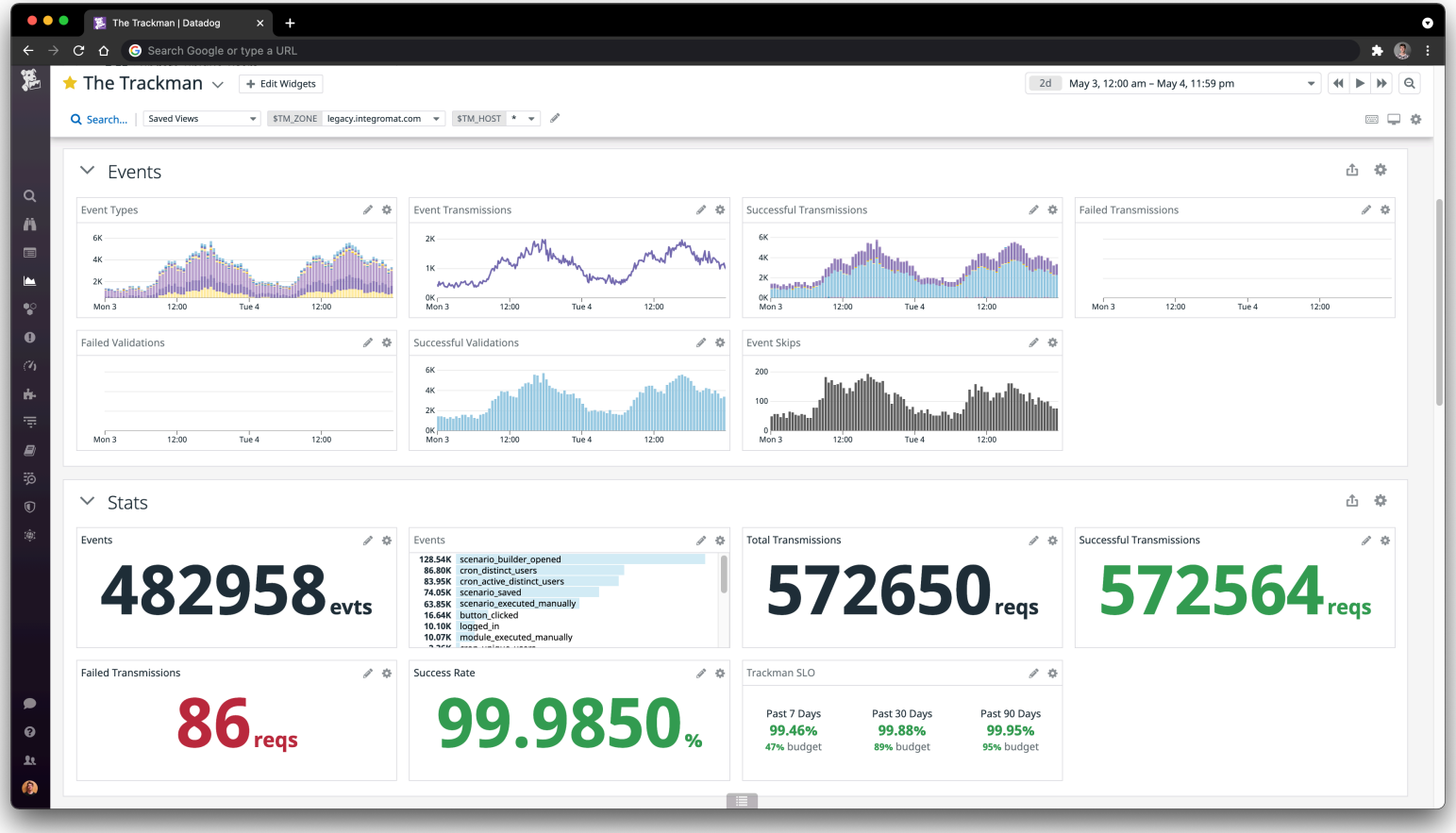
Příloha C

Ukázky z Integromatu

Zde jsou k dispozici obrázky přímo z platformy nebo z přilehlých využitých systémů.



Obrázek C.1 Scénář v prostředí Integromatu



Obrázek C.2 Monitoring Trackmana v aplikaci Datadog

Bibliografie

1. BLUE FOUNTAIN MEDIA. Why Tracking User Behavior is Essential to your Brand's Marketing Success. In: *Blue Fountain Media* [online]. 2019 [cit. 2020-11-25]. Dostupné z: `https://www.bluefountainmedia.com/blog/why-tracking-user-behavior-essential-your-brands-marketing-success`.
2. CUSTOMER.IO. *Customer.io Docs* [Online]. 2021. Dostupné také z: `https://customer.io/docs`.
3. DATADOG. *Datadog Docs* [Online]. 2021. Dostupné také z: `https://docs.datadoghq.com/`.
4. FURRER, Katie. Why It's Important to Track the Results of Your Website. In: *Thrive Web Designs* [online]. 2017 [cit. 2020-11-25]. Dostupné z: `https://www.thrivewebdesigns.com/important-track-results-website.php`.
5. MASSÉ, Mark. *REST API design rulebook: designing consistent RESTful Web Service Interfaces*. Beijing: O'Reilly, 2012. ISBN 9781449310509.
6. POKORNÝ, Jaroslav; VALENTA, Michal; ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE. *Databázové systémy*. 2020. ISBN 9788001066966. OCLC: 1200247133.
7. SOMMERVILLE, Ian. *Software engineering*. 6th ed. Harlow, England ; New York: Addison-Wesley, 2000. International computer science series. ISBN 9780201398151.
8. ŠIMEK, Patrik; KADERA, Dominik. IML. In: *Integromat Apps*. 2019. Dostupné také z: `https://docs.integromat.com/apps/other/iml`.
9. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL 12.5 Documentation* [Online]. 2020. Dostupné také z: `https://www.postgresql.org/docs/12/index.html`.
10. TYRKIEL, Kalina. Why Tracking Customer Behavior Is Important (with expert quotes). In: *Livesession* [online]. 2020 [cit. 2020-11-28]. Dostupné z: `https://livesession.io/blog/tracking-customer-behavior/`.
11. USERFLOW. *Userflow Documentation* [Online]. 2021. Dostupné také z: `https://www.userflow.com/docs`.

Obsah přiloženého média

- `README.txt` — stručný popis obsahu média
- `thesis.pdf` — text práce ve formátu PDF
- `src`
  - `js` — vybrané ukázky zdrojového kódu v jazyce JavaScript
    - `Api.js` — abstrakční vrstva na stavbu API odpovědí
    - `CronController.js` — API controller pro práci s CRONy
    - `CronRouter.js` — API router pro endpointy týkající se CRONů
    - `ImlTree.js` — vrstva nad IML umožňující jeho vyhodnocování v objektech
  - `sql` — vybrané ukázky zdrojového kódu procedur v jazyce PLpgSQL
    - `event_target_mappings_get.sql` — funkce získávající seznam dostupných mappingů
    - `events_sync.sql` — funkce starající se o synchronizaci událostí od ostatních služeb
    - `queue_next.sql` — funkce starající se o získání nové dávky událostí k odeslání
  - `thesis` — zdrojová forma práce ve formátu LaTeX