

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Zadání bakalářské práce

Student:

Libor Michálek

Studijní program:

B2647 Informační a komunikační technologie

Studijní obor:

2612R025 Informatika a výpočetní technika

Téma:

Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování:

čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Hofri s.r.o.
2. Struktura závěrečné zprávy:
 - a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
 - b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
 - c) Zvolený postup řešení zadaných úkolů.
 - d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
 - e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
 - f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

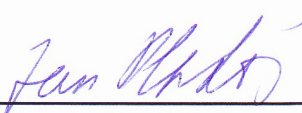
Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Marek Menšík, Ph.D.**

Konzultant bakalářské práce: Ing. Jiří Hofrichter

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020


doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry




prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 20. dubna 2020


.....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 20. dubna 2020



.....

Rád bych poděkoval za možnost absolvování individuální odborné praxe firmě Hofri s.r.o. Zejména panu Ing. Jiřímu Hofrichterovi a Ing. Janu Folwacznemu, kteří mne při vykonávání odborné praxe vedli a pomáhali s některými problémy. Dále děkuji vedoucímu mé bakalářské práce Mgr. Marku Menšíkovi, Ph.D. za konzultaci při vytváření této bakalářské práce.

Abstrakt

Tato bakalářská práce popisuje působení na odborné praxi ve firmě Hofri s.r.o., kde jsem pracoval zejména na vývoji interní aplikace pro automatizovaný denní sběr dat, která slouží k omezení a usnadnění manuální práce nutné k fakturaci klientů. V úvodu práce je popis firmy a mé pracovní zařazení. Následuje pak řešení hlavního zadaného úkolu, tvorby interní aplikace pro automatizované fakturování. Závěr obsahuje celkové hodnocení působení ve firmě, souhrn získaných vědomostí a přínosu mé individuální praxe.

Klíčová slova: TypeScript, TypeORM, PostgreSQL, AWS, Lambda, Serverless, React, Redux

Abstract

This bachelor thesis describes practicing professional practice at company Hofri s.r.o. I worked there mainly on developing internal application used for automatic data collection which are used to limit and simplify manual work needed to invoice clients. The thesis in the beginning contains company description and situating of the student at the company. In the end the thesis contains overall rating, gained knowledge and benefits of the professional practice.

Keywords: TypeScript, TypeORM, PostgreSQL, AWS, Lambda, Serverless, React, Redux

Obsah

Seznam použitých zkratek a symbolů	8
Seznam obrázků	9
Seznam tabulek	10
Seznam výpisů zdrojového kódu	11
1 Úvod	12
2 Představení společnosti	13
2.1 Odborné zaměření firmy	13
2.2 Pracovní zařazení studenta	13
3 Seznam přidělených úkolů	14
4 Vývoj Billing Service	15
4.1 Popis problému	15
4.2 Analýza	15
4.3 Návrh	16
4.4 Implementace	22
4.5 Testování a nasazení	32
4.6 Vizualizace dat	33
5 Uplatněné a scházející znalosti či dovednosti	35
5.1 Uplatněné znalosti a dovednosti získané během studia	35
5.2 Scházející znalosti či dovednosti v průběhu praxe	35
6 Závěr	36
Literatura	37
Přílohy	37
A Tabulky	38

Seznam použitých zkratk a symbolů

AWS	– Amazon Web Services
JSON	– JavaScript Object Notation
UML	– Unified Modeling Language
API	– Application Programming Interface
REST	– Representational State Transfer
AJAX	– Asynchronous JavaScript and XML
HTTP	– Hypertext Transfer Protocol
UI	– User Interface
YAML	– YAML Ain't Markup Language
CLI	– Command-line Interface

Seznam obrázků

1	Use case diagram aplikace Billing Service	16
2	UML diagram tříd reprezentující entity	17
3	Nastavení CloudWatch Rule v AWS	18
4	Schéma fungování aplikace BillingService	19
5	Diagram aktivit zpracovávání SQS zpráv	20
6	UML diagram tříd reprezentující moduly a jejich historii	24
7	UML diagram tříd reprezentující ceník a jeho vrianty	24
8	Příklad logu z Elasticsearch ve vizualizačním prostředí Kibana	26
9	Tok dat v React aplikaci s použitím knihovny Redux	33
10	Vizualizace měsíčních záznamů pomocí Recharts grafů	34
11	Vizualizace denních záznamů pomocí Recharts grafů	34

Seznam tabulek

1	Tabulka billing_daily	38
2	Tabulka billing_monthly	38
3	Tabulka eshop_configuration	39
4	Tabulka pricelist	39
5	Tabulka pricelist_variant	39

Seznam výpisů zdrojového kódu

1	PostgreSQL trigger pro vkládání historických záznamů	23
2	Ukázka zpracovaných dat o feedech ve formátu JSON	25
3	Ukázka zpracovaných dat o přístupech na web ve formátu JSON	27
4	Ukázka zpracovaných dat o předzpracování importů ve formátu JSON	28
5	Ukázka zpracovaných dat importech ve formátu JSON	29
6	Ukázka služby pro ukládání měsíčních záznamů v jazyce TypeScript	30
7	Pre-push a pre-commit Git hook	32

1 Úvod

Tato bakalářská práce pojednává o mém působení ve firmě Hofri s.r.o. v rámci individuální odborné praxe. Jejím obsahem je popis zadaného úkolu a popis jeho řešení, které spočívá ve vytvoření interní aplikace sloužící k automatizaci měsíční fakturace poskytovaných služeb.

Individuální odbornou praxi jsem si vybral za účelem nabytí nových vědomostí a získání praktických zkušeností při vývoji softwaru v týmu profesionálních vývojářů. Při vytváření zmíněné interní aplikace jsem se kromě toho podílel na rozšiřování hlavního produktu firmy, kterým je aplikace založená na frameworku Nette, architektuře MVC a napojení na externí katalog TecDoc. Produkt představuje platformní internetovou aplikaci pro obchodníky zaměřující se na prodej náhradních automobilových dílů, která spojuje dodavatelská data, interní systémy klientů a externí katalogy. Tím vytváří širokou nabídku sortimentu pro koncové zákazníky.

Druhá kapitola popisuje firmu Hofri s.r.o. a její produkt v oblasti internetového prodeje náhradních automobilových dílů a mé pracovní zařazení ve firmě. Třetí kapitola obsahuje popis zadaného úkolu a jeho rozdělení na jednotlivé části. Další kapitola líčí detailní postup řešení úkolu od analýzy a návrhu až po finální výstupy.

Závěrečná kapitola zhodnocuje celou odbornou praxi společně s výčtem znalostí a zkušeností, které mi během praxe chyběly a které jsem následně nabyl v průběhu praxe.

2 Představení společnosti

2.1 Odborné zaměření firmy

Společnost Hofri s.r.o byla založena v roce 2016, sídlí v Moravskoslezském inovačním centru v Ostravě. Firma se primárně zaměřuje na vývoj platformního řešení pro prodej automobilových dílů nesoucí název KVIKYMART.¹ Řešení aktuálně využívá napojení na katalog TecDoc, jeden z předních světových externích katalogů automobilů a náhradních dílů na světě, od firmy TecAlliance GmbH. Řešení také disponuje automatizovaným zpracováváním a importováním dat od dodavatelů náhradních dílů, které může být buď denní, nebo jednorázové. Tato data jsou párována s daty, které poskytuje TecDoc katalog a dohromady vytvářejí součást nabídky již konkrétního internetového obchodu. Uživatel má nadále možnost nastavit automatická pravidla globálních marží, sloužící k výpočtu cen pro koncové zákazníky. Kromě samotnému vystavení produktů v internetovém obchodě, slouží finální data k propojení s obchodními portály třetích stran, například Heureka nebo Google Shopping.

2.2 Pracovní zařazení studenta

Ve firmě Hofri s.r.o. jsem pracoval na pozici Full-Stack Developera, kde jsem se věnoval primárně vývoji aplikace pro automatizované fakturování služeb poskytovaných klientům, kterou jsme pojmenovali Billing Service. Aplikace je napsaná v jazyce TypeScript, následně se kompiluje do JavaScriptu a běží na v Node.js využívající serverless technologii poskytovanou Amazon Web Services. Použitý databázový systém je PostgreSQL, se kterým se pracuje na úrovni aplikace pomocí frameworku TypeORM. Nedílnou součástí, kterou bylo nutné vykonat, aby bylo možné potřebné data sbírat, bylo průběžně rozšiřovat dříve zmíněné řešení KVIKYMART, které má napsaný backend aplikace primárně v jazyce PHP, Nette frameworku. Frontendová část aplikace je napsaná v JavaScriptu, konkrétně ve frameworku ReactJS. Společnost využívá spoustu moderních technologií, jako je například balíčkovací systém NPM, verzovací systém Git. Dále Redis, Elastic, Docker, GraphQL a další.

3 Seznam přidělených úkolů

- **Seznámení se s architekturou systému a jednotlivými technologiemi**

Mým úkolem bylo se zorientovat za pomoci dílčích úkolů v celkové architektuře informačního systému, pochopit jednotlivé principy a koncepty, jako je například MVC, přístupy k psaní API, práci s používanými knihovnami, jako je například Dibi a další. Naučit se jazyky TypeScript společně s PHP a použité frameworky Nette, TypeORM, React a Redux.

Tento úkol si vyžádal asi 120 hodin práce.

- **Analýza a návrh aplikace**

Úkolem bylo analyzovat získávaná data a navrhnout pro příslušné tabulky v databázi a třídy v aplikaci, případy užití a podobně. Dále vybrat prostředí pro běh aplikace a technologie potřebné k implementaci. Podrobné rozebrání analýzy a návrhu se nachází v kapitole 4.2 a 4.3.

Tento úkol si vyžádal asi 40 hodin práce.

- **Implementace Billing Service a rozšiřování platformy KVIKYMART**

Úkolem bylo postupně implementovat v jazyce TypeScript všechny dílčí služby aplikace starající se o sběr dat a API sloužící k získávání již zpracovaných dat. Podrobněji je problematika rozebrána v kapitole 4.

Tento úkol si vyžádal asi 240 hodin práce.

- **Testování a nasazení**

Cílem přiděleného úkolu bylo ověřit nasbíraná data vůči skutečnosti, napsat pro většinu funkcionalit unit testy v testovacím frameworku Jest. Nasadit výslednou aplikaci do produkce. Popis problematiky se nachází v kapitole 4.5.

Tento úkol si vyžádal asi 60 hodin práce.

- **Vizualizace dat v platformě KVIKYMART**

Výstupem tohoto úkolu bylo, s využitím zhotoveného API, vytvořit přehled měsíčních výdajů pro klienty v JavaScriptu s využitím React frameworku společně s knihovnou Redux a knihovny pro grafy Recharts. Detailní popis s ukázkami se nachází v kapitole 4.6.

Tento úkol si vyžádal asi 80 hodin práce.

V následujících kapitolách budou postupně rozebrány jednotlivé přidělené úkoly.

4 Vývoj Billing Service

4.1 Popis problému

Měsíční fakturace klientů se skládá z několika dílčích částí: samotný pronájem produktu, licence služby TecDoc, počet přístupů na web, periodické nebo jednorázové generování feedů, aktivované moduly, předzpracování importních souborů a nakonec samotné importování dat do podoby zásob a produktů. Všechny tyto dílčí části je potřeba nacenit zvlášť. Informace k nim potřebné se dosud získávaly manuálně, což bylo časového hlediska neefektivní. Z toho důvodu vznikla aplikace pro automatizované fakturování klientů, dále jen Billing Service.

4.2 Analýza

4.2.1 Požadavky na aplikaci

Požadavky byly zadány firmou jako vytvoření Serverless aplikace v jazyce TypeScript s využitím frameworku TypeORM, která bude zajišťovat denní sběr dat z různých zdrojů a ukládat je ve vhodném formátu do databáze a každý měsíc agregovat tato data do měsíčních záznamů. Dále bude poskytovat REST API pro oba typy těchto dat, sloužící k napojení na platformu KVIKYMART, kde se budou nacházet grafy zobrazující měsíční a denní statistiky a také podrobné denní údaje v textové podobě.

TypeScript

TypeScript² je programovací jazyk s otevřeným zdrojovým kódem od firmy Microsoft. Jedná se o nastavbu jazyka JavaScript, který rozšiřuje o statické typování. Aplikace se pak následně kompilují do JavaScriptu.

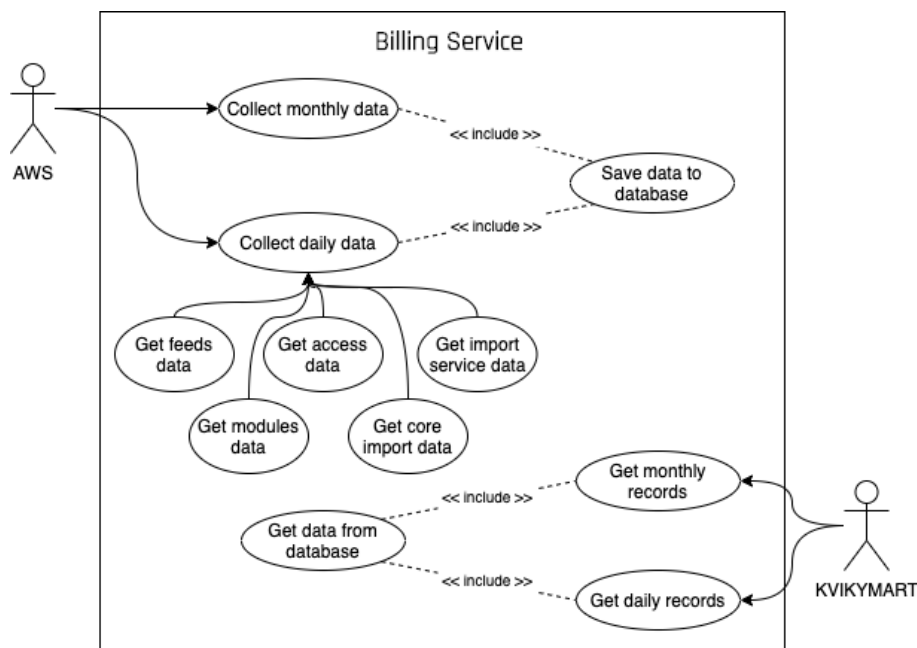
TypeORM

TypeORM³ je framework sloužící k objektově relačnímu mapování. Dokáže běžet v různých prostředích, jako například Node.js a může být použit s jazyky TypeScript a JavaScript (podporuje specifikace ES5, ES6, ES7 a ES8). Dále podporuje návrhové vzory **Active Record** a **Data Mapper**.

4.2.2 Popis uživatelů aplikace

Přímo s aplikací Billing Service pracují pouze dva externí systémy. S napojeným systémem KVIKYMART, ve které se nachází frontendová část pro vizualizaci sbíraných dat, pracují administrátoři systému. Tyto externí systémy jsou:

- **AWS CloudWatch** - Pomocí služby CloudWatch z AWS se inicializuje automatický denní či měsíční sběr dat.
- **KVIKYMART** - Platforma od firmy Hofri s.r.o. vyžívající aplikační rozhraní Billing Service, pomocí kterého vizualizuje data pro administrátory.



Obrázek 1: Use case diagram aplikace Billing Service

4.2.3 Základní objekty aplikace

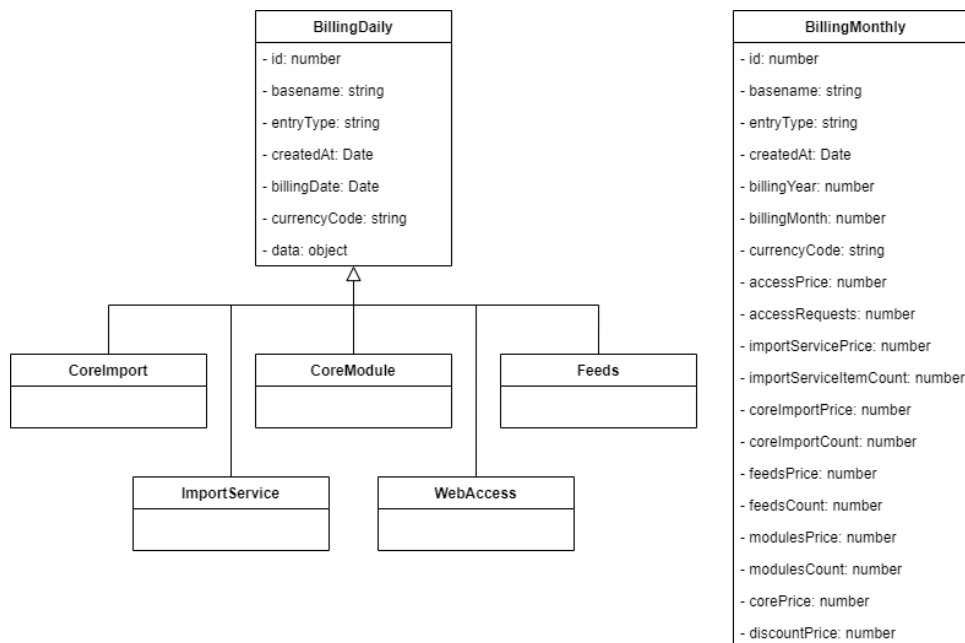
Základní jednotkou zpracování dat je **worker**. **Worker** je abstrakcí samostatné služby pro zpracování určitého typu dat. Jinak řečeno, o každý typ dat se stará jedna dílčí služba - **worker**.

Jednotkou pro výstupní rozhraní aplikace je **API**. Každé **API** je reprezentováno samostatnou **lambdou** s unikátním URL.

4.3 Návrh

Aplikace Billing Service denně sbírá různá data z několika zdrojů. O každém denním záznamu je potřeba ukládat obdobné informace: o jaký typ záznamu se jedná, kdy byl vytvořen, pro jakého

klienta a za jaký den. Z toho důvodu byl pro tyto entity použit návrhový vzor **Single Table Inheritance**,⁴ který zajišťuje, že všechny třídy v hierarchii využívají v databázi pouze jednu tabulku. Sloupec **data** představuje informace ve formátu JSON, které jsou specifické pro každý typ záznamu.



Obrázek 2: UML diagram tříd reprezentující entity

Celá aplikace běží na AWS jako Serverless Lambda. AWS Lambda dovoluje spouštět kód bez potřeby správy serverů. AWS automaticky řeší škálování podle vytíženosti aplikace a účtuje provozovateli každých 100ms běhu. Lambda může být spuštěna AWS službami jako je S3, DynamoDB, Kinesis, API Gateway, SNS, CloudWatch nebo může být součástí AWS Step Functions.

Amazon CloudWatch

Amazon CloudWatch⁵ je služba poskytující vývojářům nástroje k monitorování aplikací. Poskytuje data ve formě metrik, logů a událostí. Dokáže detekovat anomálie při běhu aplikace, vizualizovat logy, nastavovat alarmy, v jejich návaznosti vykonávat automatizované akce a spoustu dalších funkcí.

Pro tuto aplikaci je nejvhodnější použití služby Amazon CloudWatch Events, která umožňuje spouštět denně po půlnoci Lambdu, která shromažďuje a vypočítává denní výdaje jednotlivých klientů. Toto automatizované spouštění probíhá na základě cron výrazu, který je nastaven denně na 23:30 UTC času a obsahem zprávy je vždy: `{"worker": "START_DAILY_BILLING"}`. Analogicky potom platí pro generování měsíčních záznamů totéž, pouze probíhá měsíčně.

Cron výraz

Cron výraz⁶ je speciální textový formát k určení časového plánu. Každý výraz se skládá z 6 povinných polí: minuty, hodiny, den v měsíci, měsíc, den v týdnu a rok. V tomto konkrétním případě vypadá zápis takto: "30 23 * * ? *".

Summary

Schedule Cron expression

Next 10 Trigger Date(s)

1. Wed, 01 Apr 2020 23:30:00 GMT
2. Thu, 02 Apr 2020 23:30:00 GMT
3. Fri, 03 Apr 2020 23:30:00 GMT
4. Sat, 04 Apr 2020 23:30:00 GMT
5. Sun, 05 Apr 2020 23:30:00 GMT
6. Mon, 06 Apr 2020 23:30:00 GMT
7. Tue, 07 Apr 2020 23:30:00 GMT
8. Wed, 08 Apr 2020 23:30:00 GMT
9. Thu, 09 Apr 2020 23:30:00 GMT
10. Fri, 10 Apr 2020 23:30:00 GMT

Status Enabled

Description

Monitoring [Show metrics for the rule](#)

Targets

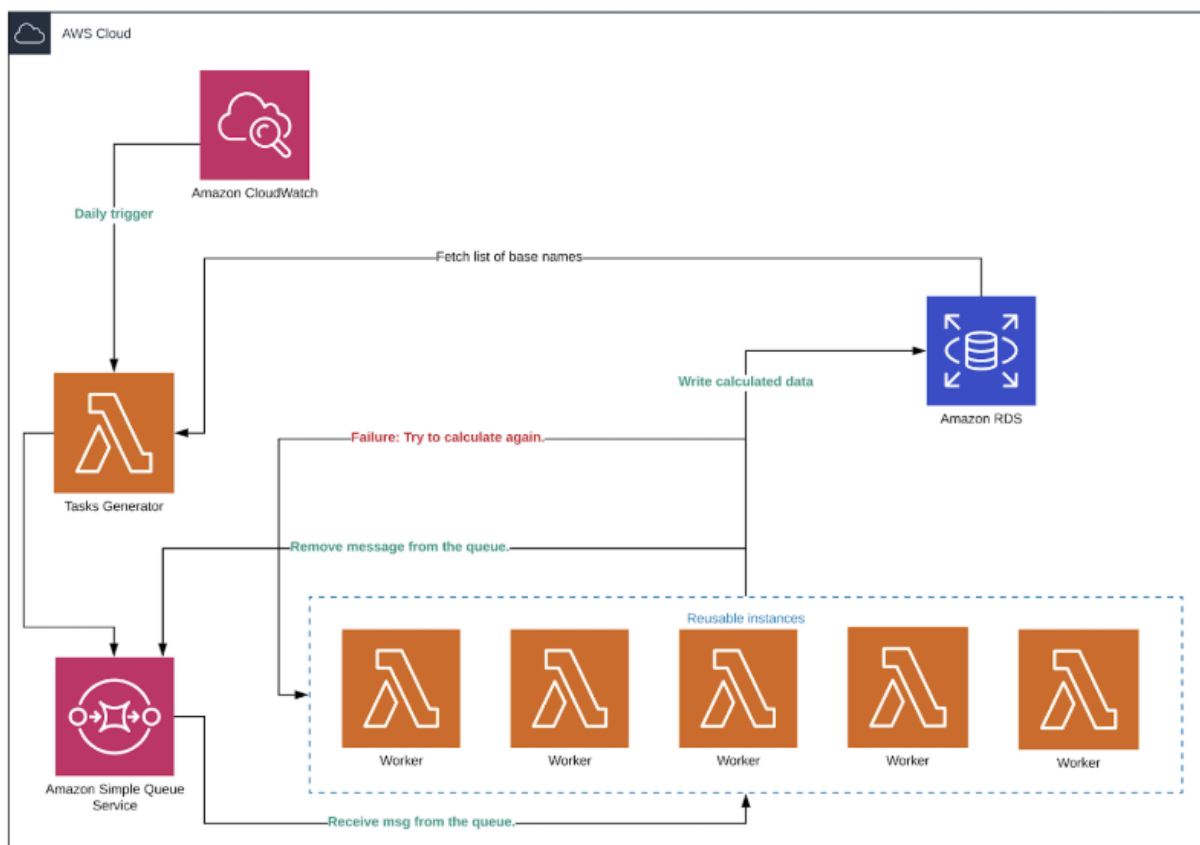
Filter: <input type="text"/>		
Type	Name	Input
Lambda function	billing-service-development-workers	Constant: {"worker": "START_DAILY_BILLING"}

Obrázek 3: Nastavení CloudWatch Rule v AWS

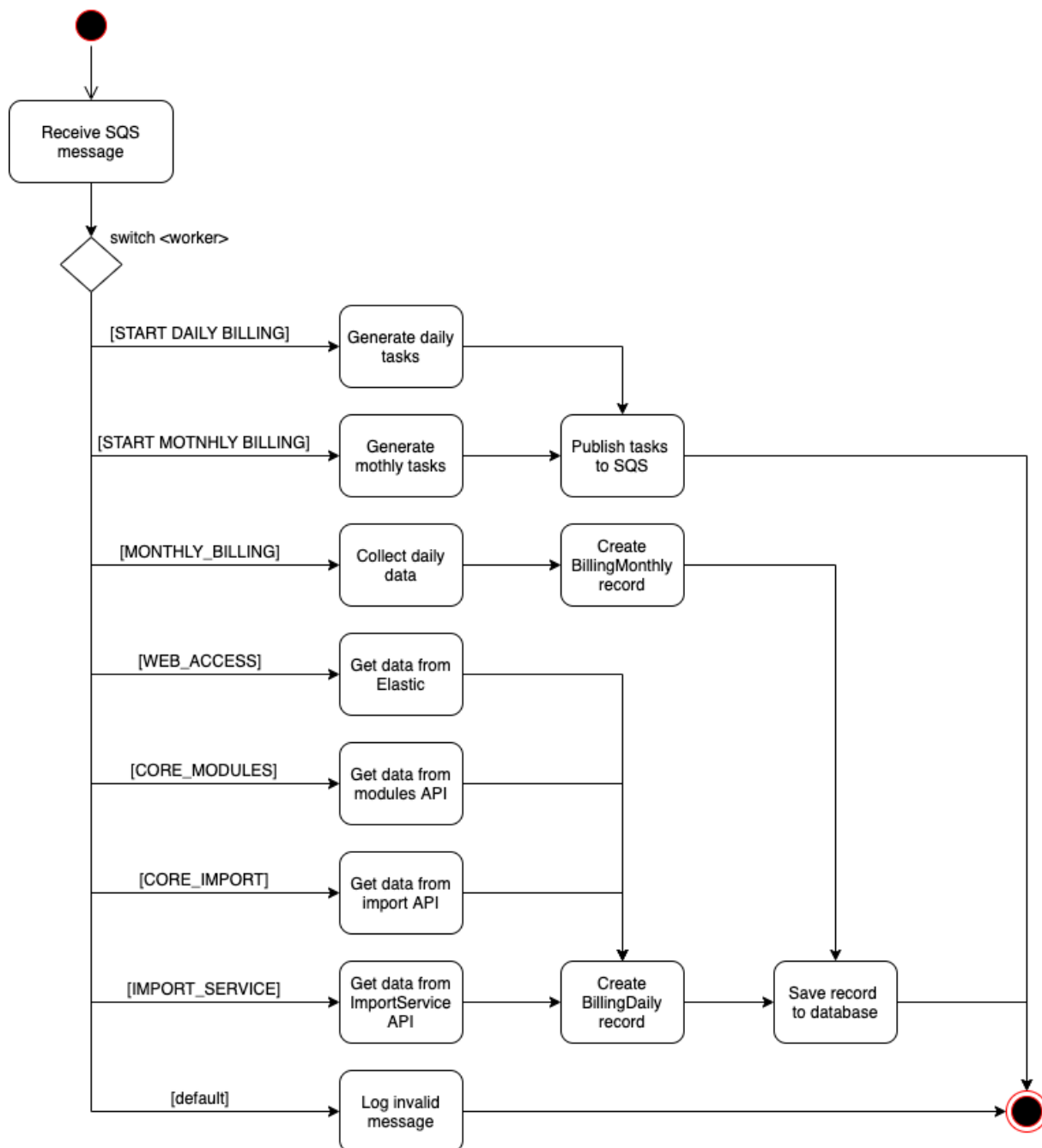
4.3.1 Wokers

Jako první je spuštěna dílčí služba aplikace Task Generator, která z PostgreSQL databáze, z tabulky `eshop_configuration` načte seznam všech provozovaných internetových obchodů firmy Hofri s.r.o. a vytvoří SQS (Simple Queue Service) zprávy, které obsahují v JSON formátu vždy: produkční a testovací URL internetového obchodu, stav ve kterém se nachází (produkční nebo předprodukční), kód měny ve které probíhá úhrada faktur klienta a typ služby, která má zprávu zpracovat, což je v rámci aplikace nazváno jako `worker`.

Zprávy jsou poté zařazeny do fronty. Po vytvoření všech SQS zpráv se spustí určitý počet znovupoužitelných instancí Lambd, které vyzvedávají zprávy z fronty a snaží se je zpracovat. Maximum takových instancí se dá ovlivnit pomocí `concurrency limit`, které je nastaveno na 5 v konfiguračním souboru aplikace `serverless.yml`. Každá zpráva ve frontě má životnost maximálně 6 hodin, nejdéle se jedna zpráva může zpracovávat 5 sekund a pokud se nepodaří úspěšně zpracovat, pokusí se ještě dvakrát opětovně spustit. V takovém případě bude v AWS CloudWatch uložena informace o neúspěšném běhu.



Obrázek 4: Schéma fungování aplikace BillingService



Obrázek 5: Diagram aktivit zpracovávání SQS zpráv

4.3.2 Lineární zápis entit

- BillingDaily(id, basename, createdAt, billingDate, currencyCode, entryType, price, data)
- BillingMonthly(id, basename, entryType, createdAt, billingYear, billingMonth, currencyCode, accessPrice, accessBilledRequests, importServicePrice, importServiceCount, coreImportPrice, coreImportChangesCount, feedsPrice, feedsCount, modulesPrice, modulesCount, corePrice, discountPrice)
- EshopConfiguration(basename, mainDomain, stagingDomain, status, domainsToRedirect, billingCurrencyCode)
- Pricelist(id, type, moduleCodename, releaseDatetime)
- PricelistVariant(id, *pricelistId*, type, languageCode, title, currencyCode, priceAmount, priceMultiplier, billingPeriod)

Podrobnější zápis entit v podobě tabulek se nachází v příloze A.

4.4 Implementace

Tato kapitola detailně popisuje implementaci aplikace, zejména dílčí části do kterých je rozdělena a následně aplikační rozhraní přes které poskytuje svá data.

4.4.1 Připojení k databázi

Na začátku běhu každé instance lamby je potřeba inicializovat připojení k databázi. Údaje k připojení poskytuje AWS Secrets Manager.⁷ Pro komunikaci s ním slouží npm balíček `aws-sdk`. Pokud nechceme obdržet údaje k produkční databázi a chceme použít například lokální databázi pro vývoj, je nutno nastavit proměnné obsahující údaje k připojení v konfiguračním souboru `.env`. Pomocí npm balíčku `dotenv` jsou pak načteny proměnné prostředí do objektu `process.env` v Node.js.

AWS Secrets Manager

AWS Secrets Manager pomáhá chránit přístupové údaje k aplikacím. Tato služba umožňuje rotaci, správu a získávání přístupových údajů k databázi, API klíčů a jiných. Uživatelé a aplikace získávající údaje využívají Secrets Manager API a tím eliminují potřebu mít citlivé údaje obsažené v rámci aplikace jako prostý text.

4.4.2 Denní záznamy

Všechny typy entit reprezentující denní záznamy dědí z třídy `BillingDaily` viz. obrázek2. Jak již bylo avizováno výše, byl pro ně použit návrhový vzor `Single Table Inheritance`, což znamená, že jsou ukládány do společné tabulky `billing_daily`. V následujících pěti kapitolách se nachází popis těchto typů entit, jak se zpracovávají v aplikaci Billing Service a jaký je jejich zdroj dat.

4.4.3 Moduly

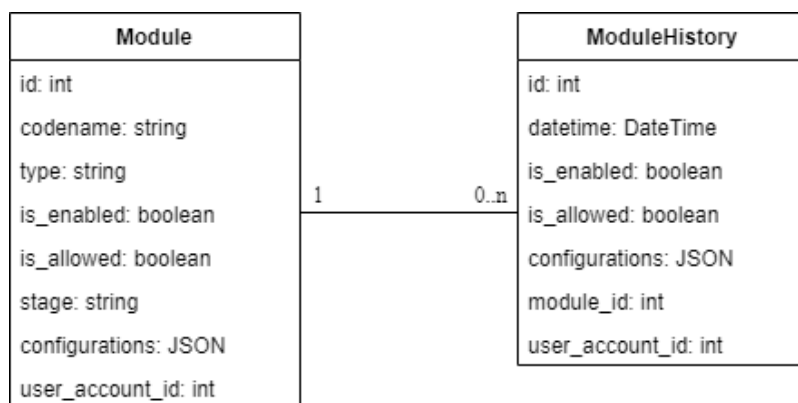
Každý internetový obchod fungující na platformě KVIKYMART udržuje stav modulů pomocí tabulky `module`. K ní bylo ovšem třeba přidat uchovávání historie v databázi. Vznikla tedy tabulka `module_history`. Dále bylo nutné rozšířit API poskytující informace o modulech tak, aby bylo schopné zrekonstruovat stav modulů každého klienta pro libovolný den.

Jelikož vkládání, upravování a odebírání záznamů tabulky `module` nevzniká vždy pouze na úrovni aplikace ale i ručně v databázi, bylo vhodné implementovat databázový trigger, který při každé změně nebo vložení nového záznamu zkopíruje informace o modulu do tabulky `module_history` a přiřadí k němu datum změny. Tím je zaručeno, že se uchová historie i při přímých změnách v databázi.

```
CREATE OR REPLACE FUNCTION module_history_record_trigger()
    RETURNS TRIGGER AS
$$
BEGIN
    INSERT INTO module_history(datetime, is_enabled, is_allowed, configurations,
        module_id, user_account_id) VALUES (NOW(), NEW.is_enabled, NEW.is_allowed, NEW.
        configurations, NEW.id, NEW.user_account_id);
    RETURN NEW;
END
$$
LANGUAGE plpgsql;
CREATE TRIGGER module_history_record_trigger
    AFTER UPDATE OR INSERT
    ON module
    FOR EACH ROW
EXECUTE PROCEDURE module_history_record_trigger();
```

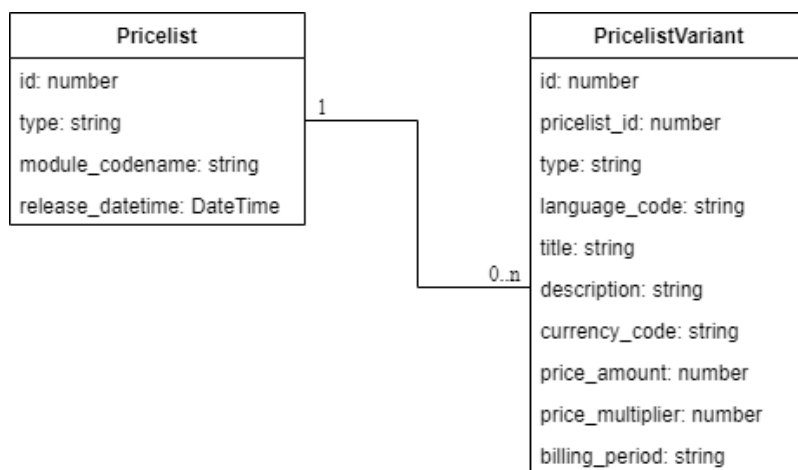
Výpis 1: PostgreSQL trigger pro vkládání historických záznamů

O modulech se v tabulce `module` evidují základní informace jako je jeho název, typ, jestli je zapnutý nebo povolený, v jakém prostředí běží (produkční nebo testovací), jeho konfigurace a který uživatel naposledy upravil jeho konfiguraci. Atributy, které se mohou často měnit, jsou pak prezentovány i v tabulce `module_history` z důvodu uchování historie. Mezi tabulkami existuje vazba 1:N.



Obrázek 6: UML diagram tříd reprezentující moduly a jejich historii

V Billing Service se následně načte pomocí interního REST API seznam aktivních modulů z předchozího dne pro konkrétního klienta a spojí se scéníkem, který je uložen v tabulkách `pricelist` a `pricelist_variant`. Mezi těmito tabulkami panuje 1:N vazba z toho důvodu, aby o každém modulu mohly být vedeny informace, jako je například cena a popis, ve více jazycích a měnách. Po získání potřebné varianty je pak určena výsledná cena pro daný modul a informace jsou zapsány do databáze ve formátu JSON.



Obrázek 7: UML diagram tříd reprezentující ceník a jeho varianty

4.4.4 Feedy

Z pohledu platformy KVIKYMART jsou feedy pouze speciální typy modulů. K získání informací o feedech každého klienta slouží stejné REST API. V aplikaci jsou však vedeny jako zvláštní záznamy z toho důvodu, že oproti modulům se o nich uchovávají komplexnější informace než o běžných modulech jako jsou například používané filtry položek nebo automatické přiřazování produktů ke kategoriím daného srovnávacího portálu ke kterému feed přísluší.

```
[
  {
    "type": "GoogleShoppingFeed",
    "filters": [
      {
        "type": "FilterExample1"
      },
      {
        "type": "FilterExample2"
      }
    ],
    "priceAmount": 1000,
    "illustrativeImagesEnabled": true,
    "categoryPredefinedAssignmentEnabled": false
  },
  {
    "type": "ZboziCzFeed",
    "filters": [],
    "priceAmount": 123,
    "illustrativeImagesEnabled": false,
    "categoryPredefinedAssignmentEnabled": false
  },
  {
    "type": "HeurekaFeed",
    "filters": [],
    "priceAmount": 3333,
    "illustrativeImagesEnabled": false,
    "categoryPredefinedAssignmentEnabled": true
  }
]
```

Výpis 2: Ukázka zpracovaných dat o feedech ve formátu JSON

4.4.5 Přístupy na web

Přístupy na web se automaticky ukládají do Elasticsearch pomocí Nginx logů. Patří sem uživatelské návštěvy, crawlery a API volání.

Nginx

Nginx⁸ je neplacený HTTP server, s otevřeným zdrojovým kódem. Je také reverzní proxy a IMAP/POP3 proxy serverem. Nginx je znám pro svůj vysoký výkon, stabilitu, bohaté funkce, jednoduchou konfiguraci a nízké vytížení zdrojů.

Elasticsearch

Elasticsearch⁹ je vyhledávací a analytický nástroj s otevřeným zdrojovým kódem pro všechny typy dat. Elasticsearch je postaven na Apache Lucene a byl vydán v roce 2010 společností Elasticsearch N.V., nyní známou jako Elastic. Je znám pro svou rychlost, škálovatelnost a jednoduché REST API. Elasticsearch je centrální komponentou sady nástrojů Elastic Stack pro analýzu, ukládání, obohacování a vizualizaci dat.

```
offset: 51,758 nginx.access.response_code: 200 nginx.access.method: GET nginx.access.user_name: -
nginx.access.http_version: 1.1 nginx.access.remote_ip_list: 172.30.1.100 nginx.access.url: /ochranny-
plech-proti-rozstrikovani-brzdovy-kotouc-nk-234757 nginx.access.referrer: -
nginx.access.remote_ip: 172.30.1.100 nginx.access.host: webapp.cz
nginx.access.http_x_forwarded_for: 172.30.1.100 nginx.access.body_sent.bytes: 12,558
```

Obrázek 8: Příklad logu z Elasticsearch ve vizualizačním prostředí Kibana

Všechny takové logy z předchozího dne od spuštění služby jsou získány ve formátu JSON pomocí REST API, kterým Elasticsearch disponuje. Následně jsou zpracovány a jejich počty rozděleny do struktury ve formátu JSON podle položky User-Agent nacházející se v hlavičce každého HTTP požadavku. Pokud v ní není obsažen žádný řetězec typu Googlebot, SeznamBot, FacebookBot, bingbot apod., jedná se o běžného uživatele, tudíž spadá do objektu s klíčem UserAccess.

```
{
  "GoogleBot": {
    "price": 0,
    "accessCount": 115982
  },
  "SeznamBot": {
    "price": 0,
    "accessCount": 3181
  },
  "OtherBots": {
    "price": 0,
    "accessCount": 33432
  },
  "UserAccess": {
    "price": 0,
    "accessCount": 30594
  },
  "totalAccess": {
    "price": 0,
    "accessCount": 187427
  }
}
```

Výpis 3: Ukázka zpracovaných dat o přístupech na web ve formátu JSON

Před samotným dotazem na Elasticsearch API je v BillingService vyhodnoceno, jestli se má pro dotaz použít produkční nebo testovací URL online obchodu. Pro rozlišení získávání příslušných objektů z API s daty o přístupech pro uživatelské přístupy a crawlery je potřeba dotaz nastavit tak, aby parametr `user_agent`, položka `device` musela nebo naopak nesměla obsahovat řetězec `"Spider"` podle toho, který typ přístupů chceme získat.

4.4.6 Předzpracování importů

Před samotným importem dat do platformy KVIKYMART je nutno soubory od dodavatelů, které běžně zabírají stovky MB, zpracovat a spojit s daty z webové služby katalogu TecDoc. O to se stará importní mikroslužba napsaná v Javě, která si uchovává data o provedených operacích a ty poskytuje ve svém API.

V Billing Service má pak nově vytvořená služba za úkol daná data načíst, transformovat do vhodného formátu, ocenit dle ceníku a výchozí měny z `eshop_configuration` pro daného klienta a uložit do databáze.

```
{
  "itemsTotal": 4320965,
  "importsList": [
    {
      "deleted": 1,
      "updated": 388377,
      "doneDate": "2020-03-13T18:00Z",
      "inserted": 54,
      "available": 185627,
      "importType": "ProductImporter",
      "itemsTotal": 389752,
      "notAvailable": 204291,
      "supplierName": "exampleSupplier",
      "priceAmountAvg": 730.6232245663169,
      "supplierCodename": "exampleSupplier",
      "withAdvancePrice": 0,
      "priceAmountUpdated": 89207,
      "priceAmountDecreased": 16,
      "priceAmountIncreased": 89191,
      "itemsProcessingTimeTotal": 81896
    }
  ]
}
```

Výpis 4: Ukázka zpracovaných dat o předzpracování importů ve formátu JSON

4.4.7 Importy

Poté, co jsou data předzpracovaná, jsou následně vyzvedávána z fronty KVIKYMART aplikací a importována. Udrží si v databázi Redis metadata utřízená podle data jejich zpracování, která obsahují informace: o jakého dodavatele se jedná, o počtu aktualizovaných, vytvořených a smazaných produktech a také o počtu zpracovaných obrázků. Tyto aplikace pak nabízí ve svém API.

Služba v Billing Service provede HTTP požadavek na endpoint každého online obchodu. Stejně jako ostatní dílčí služby pracující přímo s platformou KVIKYMART si vybírá URL podle režimu ve kterém obchod momentálně běží, což může být produkční nebo testovací. Nepovinným parametrem požadavku je parametr **date**. Protože chceme získat data vždy za předchozí den, je potřeba nastavit hodnotu tohoto parametru na datum předcházejícího dne ve formátu ISO 8601.

Data jsou následně transformována do vyhovujícího formátu a dekorována o vypočítaný celkový počet změn pro snadnější práci s daty na frontendu a nakonec uložena do databáze.

Redis

Redis¹⁰ je úložiště datových párů klíč-hodnota používané jako databáze nebo cache. Podporuje širokou škálu datových struktur a má nízkou latenci a vysokou propustnost.

```
{
  "totalChangesCount": 929,
  "coreImportsList": [
    {
      "price": 0,
      "dateTime": "2020-03-25T18:48:46+01:00",
      "imagesCount": 56,
      "supplierCodename": "Supplier1",
      "deletedProductsCount": 7,
      "updatedProductsCount": 594,
      "insertedProductsCount": 23
    }
  ]
}
```

Výpis 5: Ukázka zpracovaných dat importech ve formátu JSON

4.4.8 Měsíční záznamy

Jednou měsíčně je spuštěna služba, která si pomocí TypeORM načte všechny denní záznamy z databáze za poslední měsíc pro jednotlivé klienty a agreguje základní údaje týkající se zejména celkových cen a počtů položek všech typů denních záznamů, které mohou reprezentovat například počet feedů, modulů nebo webových přístupů a následně jej uloží do tabulky `billing_monthly`, viz Obrázek 2.

```
let billingDailyEntities: BillingDailyEntity[];
const [after, before] = this.getBetweenDates();
try {
    billingDailyEntities = await this.billingDailyRepository.find({
        where: {
            billingDate: Between(after, before),
            basename,
        },
    });
} catch (err) {
    logger.error('Failed to fetch billing daily entries.', { where: { billingDate:
        Between(after, before), basename } }, err);
}

// Zpracování záznamů
// ...

try {
    await this.billingMonthlyRepository.insert(billingMonthly);
} catch (err) {
    logger.error('Billing Monthly results has failed to insert into database.', {}, err
    );
}
```

Výpis 6: Ukázka služby pro ukládání měsíčních záznamů v jazyce TypeScript

4.4.9 Aplikační rozhraní

Aby byly data nadále využitelné pro jiné aplikace, v tomto konkrétním případě pro vizualizaci dat v administraci platformy KVIKYMART, bylo potřeba vytvořit aplikační rozhraní, které bude poskytovat podle potřeby denní nebo měsíční záznamy.

Endpointy:

- GET /monthlyBilling
- GET /dailyBilling

Jako autentifikaci ke každému HTTP požadavku na tyto endpointy je nutno v hlavičce přiložit `x-api-key`, který je generován Serverless frameworkem.

Parametry požadavku:

- `basename` - Doména internetového obchodu pro který je potřeba získat data.
- `fromDate` - Datum od kterého jsou záznamy požadovány. Tvar data musí vyhovovat normě ISO 8601.
- `toDate` - Datum do kterého jsou záznamy požadovány. Tvar data musí vyhovovat normě ISO 8601.

Pro oba endpointy platí však určitá omezení. Denních záznamů je možno v jednom požadavku vrátit maximálně 31 a měsíčních 12.

4.5 Testování a nasazení

Tato kapitola popisuje způsoby, jakými se v rámci tohoto projektu řeší ověřování funkcionality kódu, dodržování příslušného formátování kódu a nasazení aplikace do produkční prostředí.

4.5.1 Testování

Testování funkcionalit aplikace je prováděno pomocí Unit testingu. Testy pokrývají téměř veškerou funkcionalitu aplikace, od založení spojení s databází, přes API volání až po vkládání záznamů do databáze. Pro testování se využívá framework **Jest**,¹¹ určený k testování aplikací založených na JavaScriptu, který se zaměřuje na jednoduchost a obsahuje řadu předpřipravených nástrojů a funkcí usnadňující proces testování, například funkce pro mockování modulů. Testy se automaticky spouštějí při každém pokusu o akci Push ve verzovacím systému Git. Tomuto se říká Git hook z npm balíčku **husky** a jeho nastavení je specifikováno v souboru **package.json**. Druhým hookem je statická analýza kódu před každým commitem pomocí linter nástroje.

```
"husky": {
  "hooks": {
    "pre-commit": "lint-staged",
    "pre-push": "npm run test"
  }
},
```

Výpis 7: Pre-push a pre-commit Git hook

4.5.2 Nasazení

Aplikace běží jako AWS Lambda s využitím frameworku Serverless.

Serverless

Serverless¹² je framework, který umožňuje jednoduché vyvíjení, nasazování, zabezpečení, testování a monitorování serverless aplikací.

Veškerá konfigurace aplikace, týkající se zabezpečení, názvů endpointů, paměťových či časových limitů a podobně je ve formátu YAML v souboru **serverless.yml**. Deployment aplikace probíhá pomocí npm skriptu **deploy:dev** nebo **deploy:prod**, podle toho, pro které prostředí chceme kód nasadit. Tento skript spojuje spuštění testů a následně deploy pomocí Serverless frameworku. Tudíž se nemůže stát, že by byl nasazen kód bez úspěšného proběhnutí testů. V budoucnu se toto zautomatizuje pomocí Bitbucket Pipelines.

4.6 Vizualizace dat

Nasbíraná data jsou zobrazována v administraci platformy KVIKYMART na úvodní straně. Data jsou vizualizována pomocí grafů, konkrétně knihovny Recharts.¹³

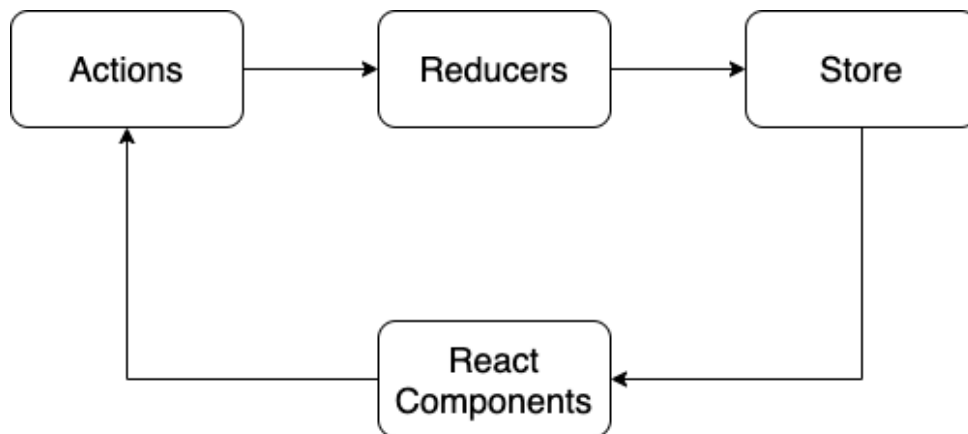
Přehled je rozdělen do tří uživatelských pohledů. Prvním je čárový graf zobrazující data za posledních 12 měsíců. Po vybrání konkrétního měsíce se zobrazí denní přehled v podobě sloupcového grafu za vybraný měsíc a nakonec při výběru konkrétního dne se zobrazí detailní výpis fakturovaných služeb v daný den v textové podobě. Veškerá API volání vztahující se k těmto akcím a překreslování UI jsou asynchronní, tudíž není třeba znovu načítat stránku. V této části aplikace je použita knihovna React a Redux.

React

React¹⁴ je JavaScriptová knihovna pro tvorbu uživatelských rozhraní. Byla vytvořena společností Facebook a využívá se pro tvorbu single-page webových aplikací nebo mobilních aplikací. Obvykle je pro jeho efektivní využití nutno použít další knihovny pro uchovávání stavu aplikace, jako je například Redux.

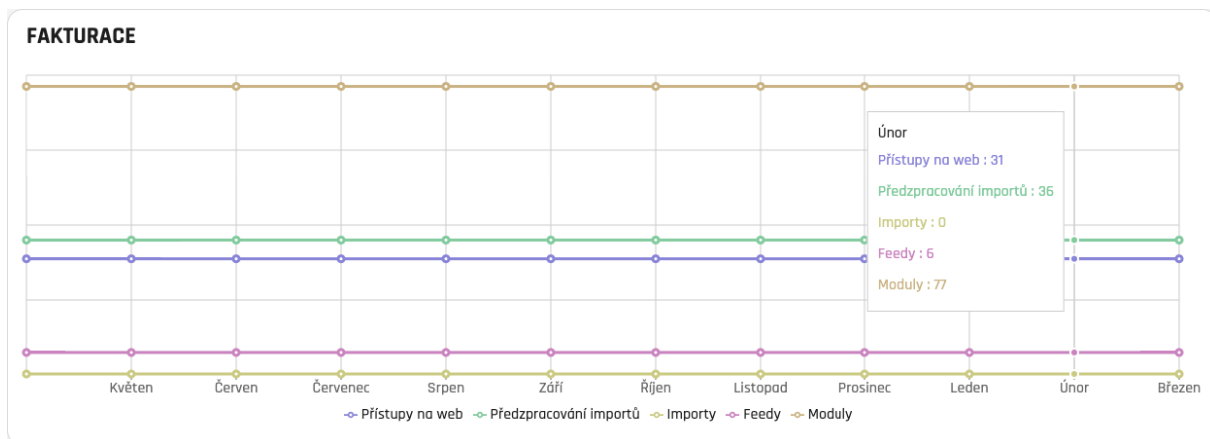
Redux

Redux¹⁵ je JavaScriptová knihovna s otevřeným kódem pro řízení stavu aplikace, která je velmi malá s jednoduchým API.

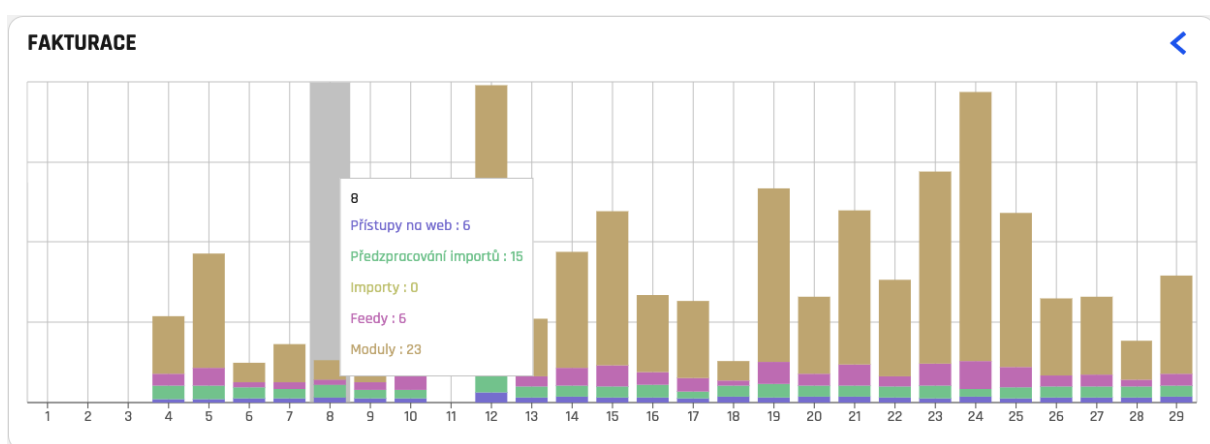


Obrázek 9: Tok dat v React aplikaci s použitím knihovny Redux

Frontendovou část tvoří 4 nové React komponenty. První je BillingsLayout, obalující ostatní funkční komponenty a řešící logiku přepínání pohledů. Při načtení stránky se zobrazí druhá komponenta MonthlyLineChart, která obsahuje čárový graf do kterého jsou načtená data z API. Na tento graf je navěšena událost, kdy se po kliknutí na konkrétní měsíc v grafu načtou denní data pro daný měsíc opět z API poskytované Billing Service. Následně se vykreslí s těmito daty další komponenta DailyBarChart, která reprezentuje tato data pomocí sloupcového grafu.



Obrázek 10: Vizualizace měsíčních záznamů pomocí Recharts grafů



Obrázek 11: Vizualizace denních záznamů pomocí Recharts grafů

Načtená data se udržují v Redux **store** a při přepínání pohledů se komponenty nepřekreslují, pokud se data uložená ve **store** nezmění. Aktuální stav pohledu v aplikaci se udržuje pomocí React Component State. Díky tomu je možno v BillingsLayout rozhodnout, která komponenta by se měla aktuálně vykreslovat. Následně má uživatel možnost vybrat konkrétní den a zobrazit si o něm detailní údaje v textové podobě a nebo se vrátit zpět na denní či měsíční přehled.

5 Uplatněné a scházející znalosti či dovednosti

Tato kapitola se zabývá shrnutím mých znalostí získaných studiem a během praxe. Taktéž se zabývá scházejícími dovednostmi během praxe.

5.1 Uplatněné znalosti a dovednosti získané během studia

Za uplatněné znalosti ze studia na praxi považuji zejména předměty Databázové a Informační Systémy a Úvod do Databázových Systémů, které poskytují potřebné znalosti z oblasti relačních databází. Dále jsem zužitkoval základy programování v jazyce JavaScript, zejména TypeScript z předmětu Vývoj Internetových Aplikací a v neposlední řadě znalosti UML a návrhových vzorů z předmětu Vývoj Informačních Systémů.

5.2 Scházející znalosti či dovednosti v průběhu praxe

Hlavní dovednost, která mi scházela při nástupu na praxi, byla týmová práce s verzovacím systémem Git. Ačkoliv jsem se s ním dostal do styku při absolvování předmětu Tvorba Aplikací pro Mobilní Zařízení II, jeho využití zde bylo velice omezené. Dále mi scházela znalost některých jazyků a většiny firmou používaných frameworků, které jsem se ale díky dobrým základům programování z univerzity dokázal snadno doučit.

6 Závěr

Výsledkem odborné praxe je aplikace, která denně automaticky sbírá a zpracovává informace potřebné k fakturaci klientů. Jednou měsíčně vytvoří z těchto záznamů jeden záznam shrnující celý měsíc a poskytuje REST API pro vizualizaci dat. Frontendovou část se podařilo dokončit pouze v její základní podobě. Není kompletní nacenění všech služeb, jelikož pro správné určení cen, je nutno pár měsíců sbírat data a následně je vyhodnotit. Nicméně na těchto částech budu nadále pracovat a snažit se je kompletně vyhotovit.

Odbornou praxi hodnotím oboustranně pozitivně. Firma má nyní na jednom místě lepší přehled o detailech služeb využívaných svými klienty a tím dokáže optimalizovat svůj ceník a poskytnout svým klientům přehled o využívaných službách. Seznámil jsem se s vývojem komplexního projektu a práce v týmu. Dále jsem získal velké množství zkušenosti s různými technologiemi a nástroji jako je například Serverless, TypeScript, TypeORM, Elasticsearch, Redis, React, Redux a spoustou dalších.

Literatura

1. *KVIKYMART* [online] [cit. 2020-03-15]. Dostupné z: <https://kvikymart.com/>.
2. *TypeScript* [online] [cit. 2020-03-15]. Dostupné z: <https://www.typescriptlang.org/>.
3. *TypeORM* [online] [cit. 2020-03-15]. Dostupné z: <https://typeorm.io/>.
4. FOWLER, Martin. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002. ISBN 0321127420.
5. *Amazon CloudWatch* [online] [cit. 2020-03-15]. Dostupné z: <https://aws.amazon.com/cloudwatch/>.
6. *Cron Expression* [online] [cit. 2020-03-15]. Dostupné z: <https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/ScheduledEvents.html>.
7. *AWS Secrets Manager* [online] [cit. 2020-03-15]. Dostupné z: <https://aws.amazon.com/secrets-manager/>.
8. *Nginx* [online] [cit. 2020-03-15]. Dostupné z: <https://www.nginx.com/resources/wiki/>.
9. *Elasticsearch* [online] [cit. 2020-03-15]. Dostupné z: <https://www.elastic.co/what-is/elasticsearch>.
10. *Redis* [online] [cit. 2020-03-15]. Dostupné z: <https://redis.io/topics/introduction/>.
11. *Jest* [online] [cit. 2020-03-15]. Dostupné z: <https://jestjs.io/>.
12. *Serverless* [online] [cit. 2020-03-15]. Dostupné z: <https://serverless.com/>.
13. *Recharts* [online] [cit. 2020-03-15]. Dostupné z: <http://recharts.org/>.
14. *React* [online] [cit. 2020-03-15]. Dostupné z: <https://reactjs.org/>.
15. *Redux* [online] [cit. 2020-03-15]. Dostupné z: <https://redux.js.org/>.
16. *AWS Lambda* [online] [cit. 2020-03-15]. Dostupné z: <https://aws.amazon.com/lambda/>.
17. GOJKO, Adzic; NIKOLA, Korac. *Running Serverless: Introduction to AWS Lambda and the Serverless Application Model*. Neuri Consulting Llp, 2019. ISBN 0993088155.
18. FREEMAN, Adam. *Essential TypeScript: From Beginner to Pro*. Apress, 2019. ISBN 148424978X.
19. *Amazon Web Services* [online] [cit. 2020-03-15]. Dostupné z: <https://aws.amazon.com/>.

A Tabulky

Název	Datový typ	Klíč	Null
id	integer	Primární	Ne
basename	text	Ne	Ne
created_at	timestamp	Ne	Ne
billing_date	timestamp	Ne	Ne
currency_code	text	Ne	Ne
entry_type	text	Ne	Ne
price	numeric(18,4)	Ne	Ne
data	jsonb	Ne	Ne

Tabulka 1: Tabulka billing_daily

Název	Datový typ	Klíč	Null
id	integer	Primární	Ne
basename	text	Ne	Ne
entry_type	text	Ne	Ne
created_at	timestamp	Ne	Ne
billing_year	smallint	Ne	Ne
billing_month	smallint	Ne	Ne
currency_code	text	Ne	Ne
access_price	numeric(18,4)	Ne	Ne
access_billed_requests	integer	Ne	Ne
import_service_price	numeric(18,4)	Ne	Ne
import_service_item_count	integer	Ne	Ne
core_import_price	numeric(18,4)	Ne	Ne
core_import_changes_count	integer	Ne	Ne
feeds_price	numeric(18,4)	Ne	Ne
feeds_count	integer	Ne	Ne
modules_price	numeric(18,4)	Ne	Ne
modules_count	integer	Ne	Ne
core_price	numeric(18,4)	Ne	Ne
discount_price	numeric(18,4)	Ne	Ne

Tabulka 2: Tabulka billing_monthly

Název	Datový typ	Klíč	Null
basename	text	Primární	Ne
main_domain	text	Ne	Ano
staging_domain	text	Ne	Ano
status	text	Ne	Ne
domains_to_redirect	jsonb	Ne	Ano
billing_currency_code	char(3)	Ne	Ne

Tabulka 3: Tabulka eshop_configuration

Název	Datový typ	Klíč	Null
id	integer	Primární	Ne
type	pricelist_type_enum	Ne	Ne
module_codename	text	Ne	Ne
release_datetime	timestamp	Ne	Ano

Tabulka 4: Tabulka pricelist

Název	Datový typ	Klíč	Null
id	integer	Primární	Ne
pricelist_id	integer	Cizí	Ne
type	pricelist_variant_type_enum	Ne	Ne
language_code	char(2)	Ne	Ano
title	text	Ne	Ano
currency_code	char(3)	Ne	Ano
price_amount	numeric(18,4)	Ne	Ano
price_multiplier	smallint	Ne	Ne
billing_period	text	Ne	Ano

Tabulka 5: Tabulka pricelist_variant