Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Návrh a realizace interního systému pro skladové zásoby

Pavel Jakl

© 2023 ČZU v Praze

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Pavel Jakl

Informatika

Název práce

Návrh a realizace interního systému pro skladové zásoby

Název anglicky

Design and implementation of an internal inventory system

Cíle práce

Hlavním cílem diplomové práce bude navrhnout a vytvořit modul informačního systému pro správu skladových zásob a jeho implementaci. Dílčím cílem bude analyzování existujících dat a navržení statistických ukazatelů, jejichž postupy budou využity pro analytickou část systému jako pravděpodobnost zmetkovosti nebo predikce objemu produktů v čase.

Metodika

Metodika diplomové práce bude založena na studiu a analýze odborných a vědeckých informačních zdrojů vztahujících se k dané problematice, a to včetně popisu jednotlivých knihoven, které budou potřebné pro realizaci informačního systému.

K návrhu systému bude využit odpovídající software, pro realizaci budou využity vhodné knihovny. Po vytvoření bude provedena analýza existujících dat a jejich interpretace vybranými statistickými uakzateli.

Na základě syntézy teoretických poznatků a výsledků v praktické části budou formulovány závěry diplomové práce.

Doporučený rozsah práce

60 stran

Klíčová slova

php, symfony, vuejs, postgresSQL, APIplatfrom, swagger, časové řady, pravděpodobnost, docker

---

Doporučené zdroje informací

Ali, Junade. Mastering PHP Design Patterns : Develop Robust and Reusable Code Using a Multitude of Design Patterns for PHP 7, Packt Publishing, Limited, 2016. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/techlib-ebooks/detail.action?docID=4699931.

BRUCKNER, Tomáš. *Tvorba informačních systémů : principy, metodiky, architektury.* Praha: Grada, 2012. ISBN 978-80-247-4153-6.

HINDLS, Richard; HRONOVÁ, Stanislava; NOVÁK, Ilja. *Analýza dat v manažerském rozhodování.* Praha: Grada, 1999. ISBN 80-7169-255-7.

Kaur, Manpreet, and Baji Shaik. PostgreSQL Development Essentials : Develop Programmatic Functions to Create Powerful Database Applications, Packt Publishing, Limited, 2016. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/techlib-ebooks/detail.action?docID=4699621.

NOVOTNÝ, Ota; POUR, Jan; SLÁNSKÝ, David; ČESKÁ SPOLEČNOST PRO SYSTÉMOVOU INTEGRACI. *Business intelligence : jak využít bohatství ve vašich datech.* Praha: Grada, 2005. ISBN 80-247-1094-3.

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

doc. Ing. Edita Šilerová, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 14. 11. 2022

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 29. 11. 2023

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Návrh a realizace interního systému pro skladové zásoby" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 29. 11. 2023 _____

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Editě Šilerové, Ph.D. za odborné vedení diplomové práce a za cenné rady při její tvorbě. Dále bych chtěl poděkovat firmě, ve které pracuji, za příležitost vytvořit danou aplikaci.

Návrh a realizace interního systému pro skladové zásoby

Abstrakt

Diplomová práce se zaměřuje na návrh a vytvoření modulu informačního systému pro správu skladových zásob. Hlavním cílem je poskytnout efektivní nástroj, který usnadní jejich správu, a to zejména prostřednictvím implementace nového modulu. Tato práce dále klade důraz na analýzu existujících dat a základní návrhy statistických ukazatelů, jako například predikci objemu produktů v čase, což je důležité pro analytickou část systému.

Z metodologického hlediska je práce postavena na studiu a analýze odborných a vědeckých informačních zdrojů vztahujících se k problematice vývoje softwaru a informačních systémů. V práci je podrobně popsáno využití různých knihoven potřebných pro realizaci navrhovaného informačního systému. Při návrhu byl využit odpovídající software a pro implementaci byly využity vhodné knihovny. Klíčovou částí je analýza existujících dat a jejich interpretace pomocí vybraných statistických ukazatelů.

Výsledky práce představují kombinaci teoretických poznatků a praktických aplikací. Na základě syntézy teoretických informací a výsledků z praktické části byly formulovány závěry, které ukazují na efektivitu a relevanci navrženého modulu pro správu skladových zásob. Tato práce tedy nabízí komplexní pohled na problematiku vývoje softwaru, interní distribuce produktů v rámci firmy a přináší nové možnosti pro jejich efektivní řízení prostřednictvím moderního informačního systému.

Klíčová slova: informační systém, analýza, vývoj software, php, Vue.js, docker,

Design and implementation of an internal inventory system

Abstract

The master thesis focuses on the design and creation of a module of an information system for inventory management. The main objective is to provide an effective tool to facilitate their management, especially through the implementation of a new module. This thesis also emphasizes the analysis of existing data and the basic design of statistical indicators, such as the prediction of product volume over time, which is important for the analytical part of the system.

From the methodological point of view, the thesis is based on the study and analysis of professional and scientific information sources related to the issues of software development and information systems. The thesis describes in detail the use of various libraries needed for the implementation of the proposed information system. Appropriate software was used in the design and suitable libraries were used for implementation. The key part is the analysis of existing data and its interpretation using selected statistical indicators.

The results of the work represent a combination of theoretical knowledge and practical applications. Based on the synthesis of the theoretical information and the results from the practical part, conclusions were formulated that show the effectiveness and relevance of the proposed module for inventory management. Thus, this thesis offers a comprehensive view of the issues of software development, internal distribution of products within the company and brings new possibilities for their effective management through a modern information system.

Keywords: information systém, software development, analytics, php, Vue.js, docker

Obsah

1 Úvod

V dnešní době dynamického technologického rozvoje a neustále se měnících firemních procesů je klíčové, aby organizace efektivně spravovaly své interní operace. Jednou z oblastí, kde je efektivní správa důležitá, je správa zásob a prodej. Přestože se zdá, že se jedná o rutinní činnost, její optimalizace může významně přispět k celkové efektivitě a snížení časové náročnosti administrátorů firmy. Tyto procesy nemusí být pouze pro velké firmy, které řeší distribuci, velké, složité sklady nebo e-shopy, může se jednat pouze o interní prodej v rámci firmy nebo správu stavu a uložení jakékoli věci.

Každá firma chce minimalizovat náklady na rutinní úkony, které jejich zaměstnanci dělají, a tak je velmi časté, že firma investuje do její optimalizace. Tato investice je na začátku vysoká, jelikož se často jedná o systém, který je připravený přesně na míru potřeby firmy nebo se jednoduše vybere již existující řešení. Vše má ale své klady a zápory a vybrat tak vhodné řešení je někdy obtížné. Počáteční investice do systému na míru je často vyšší než investice do hotové řešení, samozřejmě se najdou i výjimky, ale přináší jednu výhodou, a to je snadná údržba a případné nové vylepšení aplikace bez potřeby předání těchto informací třetí straně.

Současnost je důkazem toho, že velká část firem raději investuje do vlastního řešení, které dává smysl v celkovém kontextu a systémy na míru jsou dnes hojně rozšířené. Je ale třeba důkladně analyzovat problém a tento problém vyřešit použitím vhodných nástrojů, které budou schopné zabezpečit všechny požadavky a především, dnes jeden z nejdůležitějších, bezpečnost, a to nejen v rámci aplikace, ale především bezpečnost aplikace oproti okolním vlivům.

V mé práci se zaměřím na analýzu, návrh a základní implementaci informačního systému pro správu skladu a prodej limonád a ostatních produktů zaměstnancům ve vybrané firmě. Cílem je nahradit současný manuální a časově náročný proces, který je náchylný k chybám, moderním a efektivním řešením založeným na digitálních technologiích.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem diplomové práce bude navrhnout a vytvořit modul informačního systému pro správu skladových zásob a jeho implementaci. Dílčím cílem bude analyzování existujících dat a navržení statistických ukazatelů, jejichž postupy budou využity pro analytickou část systému jako pravděpodobnost zmetkovosti nebo predikce objemu produktů v čase.

2.2 Metodika

Metodika diplomové práce bude založena na studiu a analýze odborných a vědeckých informačních zdrojů vztahujících se k dané problematice, a to včetně popisu jednotlivých knihoven, které budou potřebné pro realizaci informačního systému.
K návrhu systému bude využit odpovídající software, pro realizaci budou využity vhodné knihovny. Po vytvoření bude provedena analýza existujících dat a jejich interpretace vybranými statistickými ukazateli.
Na základě syntézy teoretických poznatků a výsledků v praktické části budou formulovány závěry diplomové práce.

3 Teoretická východiska

3.1 Teoretický popis termínu systém a informační systém

Základní pojem, který je nutné definovat, je systém. Systém by se dal vyjádřit jako celek, který je více než jen souhrn částí, ze kterých se skládá. Celek složitých věcí mívá na rozdíl od složitých částí svou kvalitu, což je možné definovat jako svůj účel, podstatu nebo cíl či specifické účelové nebo cílové chování. (Bruckner, 2012)

Informační systém je definován jako sbírka komponent pro shromažďování, ukládání, zpracování a distribuci dat a informací. Tyto systémy jsou klíčové pro podporu obchodních cílů nebo sociotechnických systémů a nejsou samostatnými modely, ale spíše integrovanou součástí větších systémů. Zahrnují hardware, software, lidskou sílu, procesy a data. (Zwass, 2023)

3.2 Teoretická východiska existujících systémů pro skladové zásoby

3.2.1 inFlow

inFlow Inventory je moderní softwarové řešení určené k efektivní správě skladových zásob pro malé a střední podniky. Jako integrovaný systém se zaměřuje na maximalizaci efektivity skladových procesů, snížení chyb v inventarizaci a zajištění plynulého toku informací mezi odděleními. Nabízí několik modulů, jako například Inventory Control, Barcoding, reporting a mnoho dalších. (Archon Systems Inc, 2023)

Mezi jeho klíčové vlastnosti patří:

- Uživatelské rozhraní – intuitivní a uživatelsky přívětivé rozhraní
- Mobilní aplikace – jednoduchá a přehledná aplikace pro Android i iOS
- Integrace – zde nabízí integraci do více jak 95 platforem, jako je například Amazon, Magento, Shopify
- Využívání cloudu více lidmi

Celá aplikace je umístěna v cloudu, což znamená, že je potřeba mít stabilní připojení k internetu. A dále tu jsou také předplatitelské balíčky. Samotná aplikace je zdarma na vyzkoušení ve 14denní zkušební verzi a pak přichází na řadu balíčky. (Archon Systems Inc, 2023)

Balíčky dle inFlow (Archon Systems Inc, 2023) jsou celkem 4 a to:

- Podnikatel – 89$ za měsíc

- Malé podniky - 219$ za měsíc
- Střední podniky - 439$ za měsíc
- Velké podniky - 1055$ za měsíc

Všechny balíčky ještě nabízí možnost přidání různých rozšíření, které se promítnou v celkové měsíční cenně. Kromě těchto 4 balíčků je také k dispozici plán na míru, který se vyřizuje přímo s obchodníky. (Archon Systems Inc, 2023)

Kromě vlastního softwaru, firma vyrábí také hardware kompatibilní s danou aplikací, takže vše dohromady se krásně doplňuje.

3.2.1.1 SWOT analýza

Strengths (Silné stránky):

1. **Komplexní řešení**: Nabízí několik modulů, včetně ovládání zásob, čárových kódů a reportingu.
2. **Uživatelsky přívětivé rozhraní**: Intuitivní a snadno pochopitelné i pro začátečníky.
3. **Mobilní aplikace**: Podpora jak pro Android, tak pro iOS rozšiřuje možnosti používání.
4. **Rozsáhlá integrace**: Kompatibilní s více než 95 platformami, což zvyšuje jeho univerzálnost.
5. **Flexibilní cenové plány**: Různé cenové balíčky a plán na míru nabízí širokou škálu možností pro různé velikosti podniků.

Weaknesses (Slabiny):

1. **Závislost na internetu**: Jelikož je aplikace založena na cloudu, je nutné mít stále připojení k internetu.
2. **Náklady**: Přestože existuje zkušební verze, plné využití může být pro některé podniky nákladné, zejména s přidanými rozšířeními.

Opportunities (Příležitosti):

1. **Větší integrace**: Rozšiřování seznamu integrovaných platforem může přilákat více uživatelů.
2. **Rozšíření produktové řady**: Vývoj dalšího hardwaru kompatibilního s aplikací může zvýšit tržby a zákaznickou loajalitu.
3. **Lokalizace a globalizace**: Poskytování aplikace v dalších jazycích a na dalších trzích může zvýšit ziskovost.

Threats (Hrozby):

1. **Konkurence:** Trh pro softwarová řešení skladových zásob je konkurenční, s mnoha firmami nabízejícími podobné produkty.
2. **Technologické změny:** Rychlé technologické změny mohou znamenat potřebu častých aktualizací a inovací.
3. **Závislost na třetích stranách:** Pokud se některé z integrovaných platforem rozhodnou ukončit spolupráci nebo změnit své API, může to mít dopad na funkčnost inFlow.

3.2.2 Sortly

Sortly je příkladem moderního softwarového řešení, které nabízí komplexní funkce pro správu skladu. Toto řešení nejen že umožňuje podnikům sledovat a spravovat jejich zásoby v reálném čase, ale také nabízí možnosti pro automatizaci a optimalizaci skladových procesů. (Sortly Inc, 2023)

Mezi hlavní vlastnosti dle Sortly (Sortly Inc, 2023) patří:

- **Mobilní sledování:** Možnost spravovat zásoby na cestách prostřednictvím mobilní aplikace.
- **Integrace čárových kódů:** Vestavěné nástroje pro generování a skenování QR a čárových kódů.
- **Vizualizace:** Využití fotografií ve vysokém rozlišení k ověření a sledování stavu inventáře, což přispívá k lepší identifikaci položek.
- **Přizpůsobení:** Flexibilita vytváření vlastních složek a polí pro individuální potřeby podniku.

Díky těmto vlastnostem může Sortly výrazně zlepšit efektivitu skladových operací, snížit chybovost při inventarizaci a zvýšit transparentnost skladových procesů. Sortly nabízí několik balíčků, které je možno kombinovat. Jedná se například o skladový management, mapování zdrojů, prodeje, zásoby a spotřebitelé, barcoding a mnoho dalších. (Sortly Inc, 2023)

Stejně jako všechny systémy v této oblasti, má různé možnosti cenové politiky. Sortly nabízí balíček zdarma, který má jen základní funkce a slouží pouze se seznámením s prostředím a základními funkcemi. (Sortly Inc, 2023)

Sortly (Sortly Inc, 2023) dále nabízí několik balíčků:

- Pokročilý - 29$ za měsíc

- Ultra - 59$ za měsíc
- Podnikový – na domluvě přímo ze sortly

Ve všech případech se jedná buď o navýšení možnosti produktů, funkcí a dalších výhod. Vše záleží na potřebách daného podniku a předchozí analýze.

Sortly nabízí také možnost přidat vlastní pole pro specifickou činnost daného podniku a podporu pro automatizaci a optimalizaci skladových procesů.

3.2.2.1 SWOT analýza

Strengths (Silné stránky):

1. **Komplexnost:** Sortly nabízí řadu funkcí od mobilního sledování přes vizualizaci až po přizpůsobení.
2. **Flexibilita:** Možnost vytváření vlastních složek a polí pro individuální potřeby podniku.
3. **Integrace čárových kódů:** Vestavěné nástroje pro generování a skenování QR a čárových kódů, což usnadňuje sledování položek.
4. **Různé cenové balíčky:** Od základního zdarma až po podnikové řešení, což umožňuje širokému spektru podniků využít služby Sortly.

Weaknesses (Slabiny):

1. **Cena pro pokročilé funkce:** Malé podniky mohou považovat pokročilé balíčky za nákladné.
2. **Komplexnost pro nováčky:** Ačkoli je Sortly intuitivní, může jeho množství funkcí zpočátku přidělat nováčkům.
3. **Závislost na internetovém připojení:** Jelikož se jedná o online platformu, může být její funkčnost omezena v případě špatného internetového připojení.

Opportunities (Příležitosti):

1. **Expanze na nové trhy:** Vzhledem k rostoucí poptávce po digitálních nástrojích pro správu skladu může Sortly expandovat do nových regionů a odvětví.
2. **Integrace s dalšími platformami:** Možnost integrace s e-commerce platformami, účetními systémy a dalšími nástroji pro podnikání.
3. **Vývoj nových funkcí:** Reagovat na potřeby zákazníků a vytvářet nové funkce, které uspokojí konkrétní odvětvové potřeby.

Threats (Hrozby):

1. **Konkurence:** Na trhu je mnoho softwarových řešení pro správu skladu, což znamená silnou konkurenci pro Sortly.
2. **Technologické změny:** Rychlé technologické změny mohou vyžadovat časté aktualizace a investice do vývoje produktu.
3. **Zabezpečení dat:** Jelikož se jedná o online platformu, může být vystavena hrozbám spojeným s kybernetickou bezpečností.

3.3 Webová API a oddělení frontendu a backendu

V posledních letech aplikace přestávají být monolitní a více se zaměřují na architekturu a design aplikace. Existují různé typy aplikací, ať jsou to webové, mobilní, nativní či PWA aplikace. Každá z nich však musí mít datový základ a tím je databáze a jazyk, který zpracovává informace na straně serveru, například PHP, Node či další. Oddělením těchto vrstev na klientskou a serverou část dalo vzniku termínům **frontend** a **backend**. Jak název napovídá, úkolem frontendu je zajistit vizuální stránku, která se zobrazí uživateli, zajistit komunikaci s backendem, prvotní validace vstupů a přípravu http hlaviček pro komunikaci. Mezi nejpopulárnější FE jazyky řadíme například Vue.js, React, Angular, Nuxt a další.

Backend je disciplína, která zajišťuje komunikaci s databází, logické zpracování dat, autorizaci a autentifikaci. Oddělením těchto dvou „aplikací“ bylo dosaženo nového trendu pro architekturu aplikací. Vznikla tedy otázka, jakým způsobem bude probíhat komunikace mezi těmito dvěma světy a na scénu se dostává pojem webové API (application programming interface).

Webové API je sada pravidel, které definují, jakým způsobe bude probíhat komunikace mezi aplikacemi nebo zařízeními, které jsou propojené a navzájem mezi sebou komunikují. Existuje několik druhů webových API:

- Jednoduchá API
- SOAP API
- REST API
- GraphQl

3.3.1 Jednoduchá API

Tato verze je obvykle založena na CSV. Struktura dokumentu je, že na řádku je jeden záznam, který je povětšinou oddělen pomocí speciálních znaků, například svislítka. Pro jednoduchá API lze využít formát XML nebo JSON. (Sedláček, 2023)

3.3.2 SOAP API

SOAP API je standartní systém komunikačních protokolů, který umožňuje procesům využívajícím různé operační systémy (Linux, Windows) komunikovat prostřednictvím protokolu HTTP a XML. Typicky je toto rozhraní určeno pro vytváření, obnovování, aktualizaci a mazání záznamů (Malik, 2017). SOAP API má jasně definovaný standard, jelikož se jedná o protokol, a ne o architekturu a také využívá rozhraní služeb, například @WebService. (Malik, 2017)

3.3.3 REST API

REST API, které je dnes hojně využívané a oblíbené, je architektonický popis stylu komunikace mezi aplikacemi. Splňuje principy REST (representational state transfer architectural), čili přenos reprezentativního stavu. Proto někdy bývá označováno jako RESTful API. Tento styl definoval poprvé v roce 2000 Dr. Roz Fielding ve své disertační práci. Toto API je oblíbené pro svou svobodu a flexibilitu a stalo se běžnou metodou pro propojování komponent a aplikací v architektuře mikroslužeb. (IBM, 2023)

Toto API je možné naprogramovat v jakémkoli programovacím jazyce a výhodou je, že podporuje různé datové formáty. REST API tedy neukládá vývojářům žádný striktní rámec, ale je třeba dodržet následujících šest pravidla návrhu (IBM, 2023):

- Jednotné rozhraní – endpoint vrací stejná data bez ohledu na to, kolikrát je provolaný, data mají pořád stejný význam.
- Oddělení klient-server – oddělení aplikací frontendu a backendu.
- Bezestavový – zde je nutné specifikovat při každém requestu, jaká metoda je na endpoint požadována.
- Možnost uložení do paměti cache – zrychlení načítaní zdrojů dat, pokud nejsou často měněny.
- Vrstvená architektura systému – klient neví, zda komunikuje s koncovou aplikací nebo s prostředníkem.
- Kód na vyžádání – v základu API vystavuje statické zdroje dat, ale v některých případech může obsahovat spustitelný kód (Java applets). V tomto případě by se kód měl spustit pouze na vyžádání.

REST API implementuje základní CRUD operace vytvořit (**C**reate), přečíst (**R**ead), aktualizovat (**U**pdate), smazat (**D**elete). V jazyce http protokolu jim odpovídají tyto metody (Adepoju, 2020)(Sedláček, 2023):

- GET – přečíst
- POST – vytvořit
- PUT/PATCH – aktualizovat
- DELETE – smazat

3.3.4 GraphQL

GraphQL je dotazovací jazyk pro rozhraní API a běhové prostředí pro plnění těchto dotazů s existujícími daty. Poskytuje úplný a srozumitelná popis dat v rozhraní API, dává možnost vytažení přesně specifických dat, usnadňuje vývoj rozhraní API v čase a využívá výkonné vývojářské nástroje. (Adepoju, 2020)

Dle dokumentace (The GraphQL Foundation, 2023) je GrapQL přesně taková, jak jej specifikovali. Při vyžádání konkrétního dotazu se vrátí přesná odpověď na to, co bylo zadáno. API vrací vždy předvídatelné výsledky a aplikace, které využívají toto API jsou rychlé a stabilní, jelikož kontrola dat probíhá na straně aplikace a nikoli na straně serveru. Další výhodou je relační spojení výsledků, které se vrací z API. Všechna potřebná data jsou získána v jediném požadavku, na rozdíl od REST API, kde je potřeba dotázat se na více URL adres pro kompletaci výsledku.

Rozhraní GraphQL jsou uspořádána podle typů a polí, nikoli podle endpointů. Lze tedy přistupovat jen k jedinému endpointu a odsud si tahat veškerá data. Deklarací typů je zajištěno, že aplikace se bude dotazovat pouze na to, co je možné a při neúspěchu API poskytne jasné a srozumitelné chybové hlášky. Výhoda je, že se zde nemusí využívat ruční parsování kódu, jelikož jsou předem definované typy pro konkrétní dotaz. (Adepoju, 2020)

Pro přesné mapování rozhraní API slouží výkonné vývojářské nástroje, například GraphiQL, který využívá systém typů konkrétního API a umožnuje tak předcházet potencionálním problémům ještě před odesláním dotazu do rozhraní API (Foundation, 2023).

Při přidáních nových polí a typů se neovlivňují dotazy, které jsou již definovány. Díky využití jediné vyvíjející se verze poskytuje rozhraní GraphQL aplikacím nepřetržitý přístup k novým funkcím a podporuje čistší a lépe udržovatelný kód. (The GraphQL Foundation, 2023)

```
[
  {
    id: 1,
    name: "Fikayo Adepoju",
    email: "fik4christ@yahoo.com",
    posts: [
      {
        id: 1,
        title: "Debugging an Ionic Android App Using Chrome Dev Tools",
        published: true,
        link:
          "https://medium.com/@coderonfleek/debugging-an-ionic-android-app-usin
g-chrome-dev-tools-6e139b79e8d2",
        author: 1
      },
      {
        id: 2,
        title: "Hosting a Laravel Application on Azure Web App",
        published: true,
        link:
          "https://medium.com/@coderonfleek/hosting-a-laravel-application-on-az
ure-web-app-b55e12514c46",
        author: 1
      }
     ]
  },
  {
    id: 3,
    name: "Jane Paul",
    email: "jane@company.com",
    posts: []
  }
];
```

Obrázek 1 Příklad odpovědi Graphql (The GraphQL Foundation, 2023)

3.4 Teoretický popis definovaných technologií

3.4.1 PHP

PHP, jak je známe dnes, je nástupcem produktu s názvem PHP/FI. Vytvořil jej v roce 1994 Rasmus Lerdorf. První verze PHP byla jednoduchým souborem binárních dat Common Gateway Interface (CGI) napsaných v programovacím jazyce C. Původně byl tento soubor skriptů používán k sledování návštěvnosti Rasmusova online životopisu a byl nazván "Nástroje pro osobní domovskou stránku", což se často zkracovalo na "PHP Nástroje". Postupem času se objevila potřeba rozšířit jeho funkcionalitu, a tak Rasmus PHP Nástroje

přepracoval, vytvořil rozsáhlejší implementaci. Toto nové provedení bylo schopno komunikovat s databázemi a umožňovalo vytváření jednoduchých dynamických webových aplikací, jako jsou návštěvní knihy. V červnu 1995 zveřejnil Rasmus zdrojový kód PHP Nástrojů, což umožnilo vývojářům jeho využívání a úpravy podle vlastních potřeb. (The Php Group, c2001-2023)

V září téhož roku Rasmus PHP rozšířil a na krátkou dobu opustil název PHP. Nová verze byla označena jako FI (což je zkratka pro "Forms Interpreter") a zahrnovala základní funkce PHP, jak jej známe dnes. Syntaxe byla podobná Perlu, ale byla jednodušší a méně konzistentní. V říjnu 1995 Rasmus představil kompletně přepracovanou verzi kódu s návratem k názvu PHP, tentokrát jako "Stavebnice osobní domovské stránky". Byl to první release, který nabízel pokročilé skriptovací rozhraní. Jazyk byl navržen tak, aby strukturně připomínal jazyk C, což usnadňovalo přechod vývojářům obeznámeným s C, Perl a podobnými jazyky. (The Php Group, c2001-2023)

V dubnu 1996 Rasmus představil novou verzi s názvem PHP/FI, kombinací názvů předchozích verzí. Tato verze začala transformovat PHP z nástroje na plnohodnotný programovací jazyk. Obsahovala vestavěnou podporu databází DBM, mSQL a Postgres95, cookies, podporu uživatelsky definovaných funkcí a mnoho dalšího. V červnu bylo PHP/FI označeno jako verze 2.0. V listopadu 1997, když se již blížil konec beta verze, byl základní analyzátor kódu kompletně přepsán. (The Php Group, c2001-2023)

I když vývoj PHP/FI nebyl dlouhý, jeho popularita na poli webového vývoje stále rostla. V letech 1997 a 1998 měl PHP/FI tisíce uživatelů po celém světě. Podle průzkumu společnosti Netcraft z května 1998 mělo téměř 60 000 domén v hlavičkách informaci o "PHP", což naznačovalo, že hostitelský server měl PHP nainstalováno. Toto číslo představovalo přibližně 1 % všech domén na internetu v té době. Ačkoli tyto údaje byly pozoruhodné, další vývoj PHP/FI byl omezen, protože i když se na něm podílelo několik vývojářů, stále to byl projekt vedený jednotlivcem. (The Php Group, c2001-2023)

3.4.2 Composer

Composer je nástroj, který se stará o správu závislostí v jazyce PHP. Umožňuje deklaraci knihoven, na kterých je projekt závislí a jejich aktualizaci nebo instalaci. Composer spravuje

balíčky na úrovni báze jednotlivých projektů a knihovny instaluje do adresáře (vendor) uvnitř projektu. Toto chování je výchozím stavem. Pro potřeby je ale možné vytvořit globální projekt pomocí příkazu global. (Composer, 2023)

Pro bezpečné fungování composeru je důležité dodržet systémové požadavky. V nejnovější verzi composer vyžaduje PHP 7.2.5, verze 2.2.x stále nabízí podporu pro PHP 5.3.2+, pokud je to starší projekt. Při jeho instalaci je také nutné nastavit některé nastavení PHP a příznaků kompilace. (Composer, 2023)

3.4.2.1 Instalace

Composer je dostupný pro všechny distribuce operačních systémů – Linux, Unix, MacOS a Windows. Jediné, co se liší je způsob instalace na konkrétní distribuci. Composer se instaluje lokálně přímo do projektu nebo globálně, kde je Composer spustitelný pro celý systém. (Composer, 2023)

Instalace pro Unix, Linux a MacOS je prováděna pomocí instalačního programu composer.phar. Formát PHAR je archivní formát pro PHP, který se dá spouštět i z příkazového řádku. Při instalaci do konkrétní složky se u příkazu definuje konkrétní adresář a název, to je vykonáno pomocí příkazů --install-dir a --filename. Dle dokumentace (Composer, 2023) příkaz tedy vypadá následovně:

- php composer-setup.php --install-dir=bin --filename=composer
- bin/run composer – spustí composer

Při instalaci globálně, je Composer umístěn do složky, která je součástí PATH cesty. V unixových systémech lze Composer udělat samostatně spustitelným a není třeba přímého použití interpretu PHP. Pokud je potřeba instalace pouze pro konkrétního uživatele a nevyžadují se práva roota, lze využít ~/.local/bin, který je v některých linuxových distribucích k dispozici ve výchozím nastavení (Composer, 2023).

Při instalaci na systém Windows je k dispozici instalační program nebo manualní instalace. Při instalačním souboru je stažen soubor Composer-Setup.exe a instalační program nainstaluje nejnovější verzi Composeru, upraví PATH cestu a lze jej volat z jakékoli cesty v příkazové řádce. (Composer, 2023)

Dle dokumentace (Composer, 2023), je při manuální instalaci třeba přepnout do PATH adresáře a spustit instalační program, kdy je stažen souboru composer.phar. Vedle souboru je nutné vytvořit ještě soubor composer.bat. Při přidání adresáře do prostředí PATH

bude Composer přístupný. Úspěšnou instalaci a aktuální verzi Composeru lze ověřit pomocí následujících příkazů. (Composer, 2023)

```
C:\bin> echo @php "%~dp0composer.phar" %*>composer.bat
```

Obrázek 2 Instalace composer pomocí cmd.exe (Composer, 2023)

Composer lze také dokerizovat. Je to veřejný docker kontejner. Lze přidat do existujícího Dockerfilu a zkopírovat tak binární soubor z pre-buildu (Composer, 2023).

3.4.3 Symfony

Symfony je jedním z hlavních PHP frameworků pod hlavičkou licence MIT, sloužící pro tvorbu webových aplikací, systémů, portálů a jakýchkoli webových aplikací.
Instalace symfony je možná několika způsoby. Jedním z nich je instalace globálního devtoolu Symfony CLI, který také slouží k řízení buildů a běhu aplikace. (Symfony™, [2023])

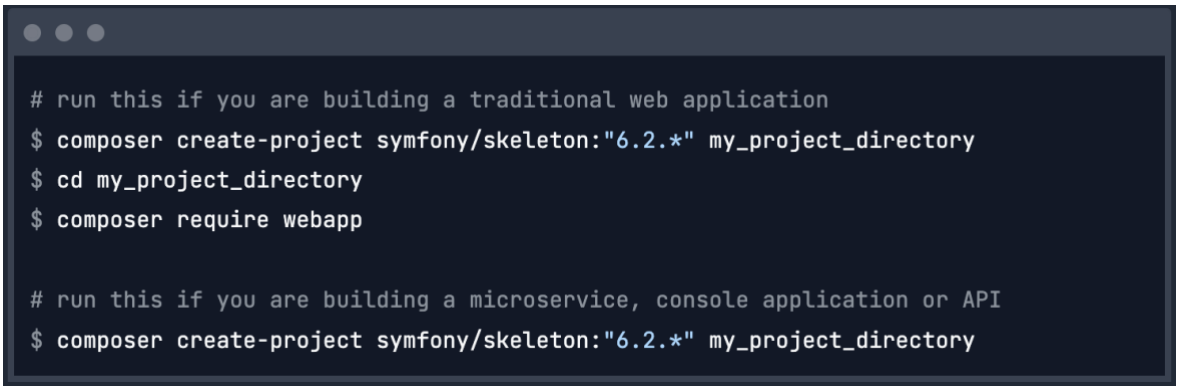
```
# run this if you are building a traditional web application
$ symfony new my_project_directory --version="6.2.*" --webapp

# run this if you are building a microservice, console application or API
$ symfony new my_project_directory --version="6.2.*"
```

Obrázek 3 Instalační skript Symfony (Symfony™, [2023])

V dokumentaci (Symfony) Symfony CLI umožňuje instalaci dvou verzí symfony

- Webapp – v této verzi se instaluje plná struktura složek včetně pohledů
- Microservice/api – zde se instalují pouze nezbytné složky, které jsou potřeba k vytvoření API, console application nebo microservices

Další možnost instalace je přes composer. I zde je možnost instalace plné nebo částečné verze symfony. (Symfony™, [2023])



Obrázek 4 Poinstalační skripty Symfony (Symfony™, [2023])

3.4.4 Docker

Docker je nástroj pro vývoj, odesílání a spouštění aplikací. Jeho hlavní výhodou je, že odděluje aplikaci od infrastruktury. Pomocí využití metodik Dockeru přichází rychlejší nasazování, testování a odesílání kódu a dochází tak ke zkrácení času mezi tvorbou aplikace a nasazením do produkce. (Docker Inc., c2013–2023)

3.4.4.1 Architektura

Základní architektura Dockeru dle (Docker Inc., c2013–2023) je tvořena třemi komponentami:

- Docker client
- Docker deamon
- Docker compose

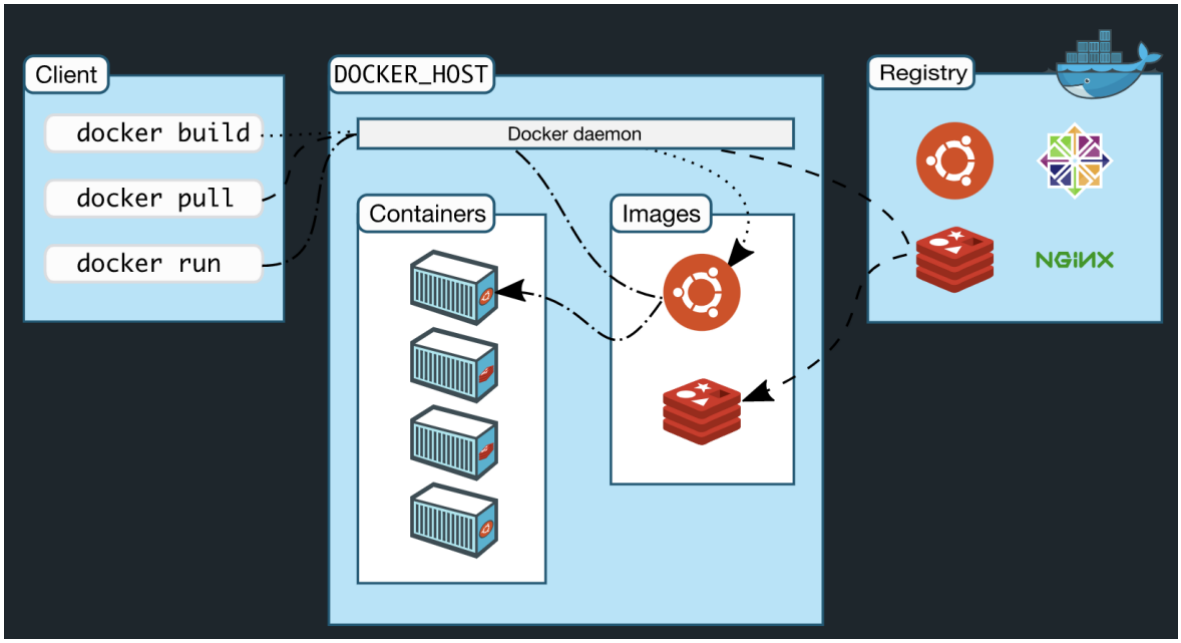
Docker využívá technologii client-server. Docker Deamon, který sestavuje, spouští a distribuuje kontejnery, které se vytváří, komunikuje s Docker client pomocí REST API, UNIX socketů nebo síťové rozhraní. Tyto dva prvky mohou běžet na stejném systému, ale je zde i možnost připojit Docker client ke vzdálenému deamonu. Docker compose, což je další klient Dockeru, umožňuje práci s aplikacemi, které jsou tvořeny ze sady kontejnerů.(Docker Inc., c2013–2023)

Při práci s Docker se vytváří objekty. Mezi tyto objekty patří například obrazy, sítě, svazky, plug-iny a další objekty. (Docker Inc., c2013–2023)

Obrazy jsou pouze pro čtení a slouží jako definice vytvoření nového kontejneru. Často je však obraz založen na jiném obrazu s určitými úpravami. Obrazy jsou vytvářeny dvěma způsoby. Jedním z nich je vytvoření osobou, která vytváří konfigurační soubor,

Dockerfile a druhý způsob je využití již existujících obrazů, které byly vytvořeny někým jiným. (Docker Inc., c2013–2023)

Při přepisu Dockerfilu a znovu zkomponování obrazu se mění pouze ty části obrazu, které se změnli. Proto je Docker ve srovnání oproti jiným virtualizačním technologiím rychlejší a malý.



Obrázek 5 Architektura Docker (Docker Inc., c2013–2023)

Kontejner je spustitelná instance obrazu. A jako k instanci se k ní lze chovat. Může být spuštěna, zastavena, přesunuta nebo smazána. Ke kontejneru lze také připojit úložiště a může být také vytvořen obraz na základě konkrétního stavu kontejneru. Ve svém výchozím stavu je kontejner dobře izolován od okolí, tedy od ostatních kontejnerů a svého hostitelského počítače. Kontejner může být ovlivněn izolovaností sítě, úložištěm, jinými základními subsystému kontejnery nebo hostitelským počítačem. Každý kontejner je jasně definován svou bitovou kopií a možnostmi, které byly poskytnuty při jeho vytvoření či spuštění. Pokud je kontejner odstraněn, jsou odebrány všechny jeho stavy, které jsou uloženy mimo trvalé úložiště. Při opětovném vytvoření kontejneru je vytvořen tak, jak je definován v Dockerfilu. (Docker Inc., c2013–2023)

3.4.5 Doctrine

Doctrine je kolekce projektů pro PHP, kdy každý projekt může být využitý v projektu samostatně. Doctrine se řadí mezi ORM (Object relational mapper) pro PHP v minimální

verzi PHP 7.1+, které poskytuje transparentní persistenci pro PHP objekty. V jádru je využíván návrhový vzor Data Mapper, který má za úkol plně oddělit doménovou logiku od perzistence v relačním systému databází.

Hlavní výhodou verze Doctrine je fakt, že soustředěnost se dává směrem na objektově orientovanou logiku a perzistence objektů je zde druhotná. Dochází tedy k oddělení entit a perzistencí.

3.4.5.1 Entity

Entity jsou objekty PHP, které lze jednoznačně rozlišit pomocí unikátního identifikátoru nebo primárního klíče. Entity nemusí rozšiřovat žádnou abstraktní třídu nebo interface. Entita jako taková obsahuje perzistentní vlastnosti, což jsou instance entity, které se ukládají do databáze a čtení těchto instancí je zajišťováno pomocí možnosti mapování dat v Doctrine. Entity by neměly být konečné, a to ani pro čtení. Naopak mohou obsahovat konečné metody nebo vlastnosti pouze pro čtení. (Doctrine, [2023])

Entity jsou v Doctrine ORM hlavní stavební kámen a jsou rozděleny do dvou kategorií. **Anemické entity** – tyto entity mají definované gettery a settery, což znamená definování čtení (getter) a aktualizace (setter). (Doctrine, [2023])

```php
<?php
class User
{
    private $username;
    private $passwordHash;
    private $bans;

    public function getUsername(): string
    {
        return $this->username;
    }

    public function setUsername(string $username): void
    {
        $this->username = $username;
    }

    public function getPasswordHash(): string
    {
        return $this->passwordHash;
    }

    public function setPasswordHash(string $passwordHash): void
    {
        $this->passwordHash = $passwordHash;
    }

    public function getBans(): array
    {
        return $this->bans;
    }
```

Obrázek 6 Anemické entity (Doctrine, [2023])

Bohaté entity – tyto entity mají tzv. mutátory a DTO, zde jsou definovány metody, které reprezentují chování dané domény. (Doctrine, [2023])

```php
<?php
class User
{
    private $banned;
    private $username;
    private $passwordHash;
    private $bans;

    public function toNickname(): string
    {
        return $this->username;
    }

    public function authenticate(string $password, callable $checkHash): bool
    {
        return $checkHash($password, $this->passwordHash) && ! $this->hasActiveBans();
    }

    public function changePassword(string $password, callable $hash): void
    {
        $this->passwordHash = $hash($password);
    }

    public function ban(\DateInterval $duration): void
    {
        assert($duration->invert !== 1);

        $this->bans[] = new Ban($this);
    }
}
```

Obrázek 7 Bohaté entity (Doctrine, [2023])

Dalším krokem pro perzistence entit je popsání jejich datové struktury. Doctrine využívá jazyk metadat. Ten popisuje, jak mají být entity, jejich vlastnosti a reference ukládány a jaká omezení se na ně vztahuje. Jazyk metadat nabízí tři formy zápisu pomocí **atributů**, **anotací** nebo **XML**. Nad každou vlastností entity je definováno její chování a omezení, které se má následně mapovat. Pomocí metadat je tedy nadefinováno například ID entity, které má být zároveň primární klíč, má vlastnost AI (auto incremental), je to generovaná hodnota, tudíž pro něj není vytvořen v případě anemické entity setter a jeho definovaný datový typ je číslo (integer). (Doctrine, [2023])

```php
#[ORM\Id]
#[ORM\GeneratedValue]
#[ORM\Column(type: 'integer')]
private int|null $id = null;
```

Obrázek 8 Specifikace pole v entitě (Doctrine, [2023])

3.4.6 API Platform

API Platfrom je sada nástrojů pod hlavičkou MIT licence, která slouží pro vytváření webových projektů. Jedná se v podstatě o Framework, který se využívá pro API-first projekty postavené nad komponentami Symfony. Veškeré balíčky, které tvoří tento framework lze využívat samostatně. (Dunglas, c2023)
Knihovna obsahuje PHP (Core) pro vytváření hypermediální nebo GraphQL webových rozhraní API, které podporují přední průmyslové standardy jako jsou JSON-LD + Hydra, OpenAPI, Json. Zároveň také obsahuje integraci Dockeru a Kubernetes definice pro vývoj a nasazení v cloudu. (Dunglas, c2023)

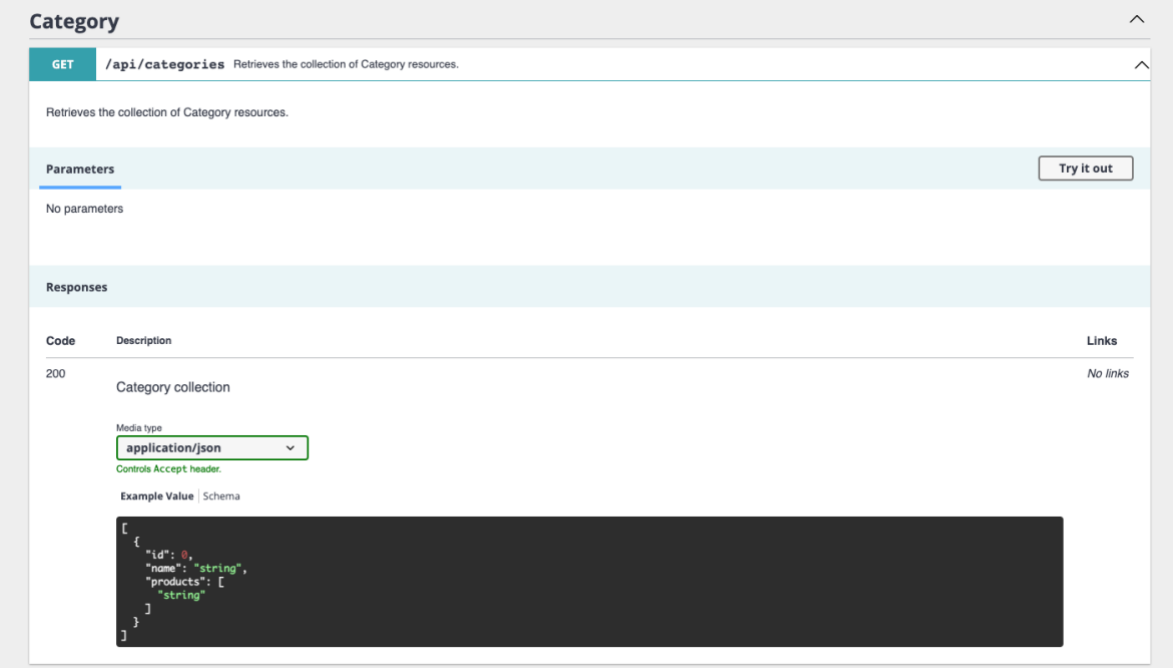
Platforma používá modelové třídy k vystavení a zdokumentování webového rozhraní API s řadou vestavěných funkcí. Mezi ně například patří:

- CRUD (vytváření, čtení, aktualizace, smazání)
- Ověřování dat
- Stránkování
- Filtrování dat
- Třídění
- Podpora GraphQL
- Integrace strojově čitelné dokumentace a uživatelské rozhraní (Swagger UI/OpenAPI, HAL)
- Autentizace – HTTP, cookies, JWT tokeny a oAuth
- CORS hlavičky

Instalace je možná několika způsoby, například využitím docker projektu, nebo stažením distribuce přímo do projektu. Při stažení platformy jako závislého balíčku do skeletonu aplikace je využíváno **composeru.**

```
symfony composer require api
```

API Platform vystavuje popis rozhraní API ve formátu **OpenAPI**. Integruje také uživatelské rozhraní Swagger, které slouží k lepší orientaci v dokumentaci OpenAPI. Prostředí se skládá z modelů, které reprezentují databázové entity a z operací, které se vykonávají na konkrétním endpointu. Jednotlivé operace lze rozkliknout a přímo z rozhraní lze posílat dotazy na rozhraní API.



Obrázek 9 Api dokumentace pomocí Swagger (vlastní zdroj)

Každá metoda má daný formát podle specifikace OpenAPI. Obsahuje popis konkrétní metody, parametry a výsledný formát odpovědi, který bude zaslán rozhraním API po úspěšném dotazu. Lze vybrat formát odpovědi v závislosti na specifikovaném formátu uvnitř API Platformy.

3.4.6.1 Formáty

API Platforma podporuje mnoho formátů, které jsou definovány globálně pomocí .yaml souborů v konfigurační složce aplikace nebo lokálně pro jednotlivé CRUD operace. Nabízí také rozlišení jednotlivých typů odpovědí. (Dunglas, c2023)

Formát	Jméno formátu	MIME typ	Kompatibilita zpět
JSON-LD	jsonld	application/ld+json	ano
GraphQL	n/a	n/a	ano
JSON:API	jsonapi	application/vnd.api+json	ano
HAL	jsonhal	application/hal+json	ano
YAML	yaml	application/x-yaml	ne
CSV	csv	text/csv	ne
HTML	html	text/html	ne
XML	xml	application/xml text/xml	ne
JSON	json	application/json	ne

Tabulka 1 Podporované formáty (Dunglas, c2023)

Lze nakonfigurovat odlišný formát pro úspěšné dotazy, errory nebo pro specifické modely a operace uvnitř modelů. (Dunglas, c2023)

Obrázek 10 Formáty pro Api Platform v kódu (vlastní zdroj)

3.4.7 PostgreSql

Projekt PostgreSQL byl zahájen na Kalifornské univerzitě v Berkeley (UCB), a od té doby byl rozvíjen komunitou, což vyvrcholilo v roce 1996. PostgreSQL je aktivně vyvíjen komunitou PostgreSQL a také různými univerzitami. (Juba, 2019)

Následující časové úseky mapují historii vývoje PostgreSQL dle (Juba, 2019):

- 1977-1985, projekt Ingres: Michael Stonebraker vytvořil relační databázový systém (RDBMS) založený na formálním relačním modelu.
- 1986-1994, Postgres: Michael Stonebraker vytvořil Postgres za účelem podpory komplexních datových typů a objektově relačního modelu.
- 1995, Postgres95: Andrew Yu a Jolly Chen upravili Postgresový dotazovací jazyk PostQUEL tak, aby zahrnoval rozšířenou podmnožinu SQL.
- 1996, PostgreSQL: Několik vývojářů věnovalo velké úsilí a čas na stabilizaci Postgres95. První open source verze byla uvolněna 29. ledna 1997. S příchodem nových funkcí, vylepšení a díky tomu, že se jedná o open source projekt, byl název Postgres95 změněn na PostgreSQL.
- PostgreSQL začal u verze 6 s velmi silným výchozím bodem, díky několikaletému výzkumu a vývoji. Jako open source projekt s velmi dobrou pověstí, přilákal PostgreSQL stovky vývojářů. Aktuálně má PostgreSQL nesčetné množství rozšíření a velmi aktivní komunitu.

PostgreSQL nabízí množství funkcí, které jsou atraktivní pro vývojáře, administrátory, architekty a firmy.

3.4.7.1 Obchodní výhody PostgreSQL

PostgreSQL dle (Juba, 2019) je svobodný software s otevřeným zdrojovým kódem (OSS). Je distribuován pod licencí PostgreSQL, která je vysoce permisivní, a PostgreSQL není předmětem monopolů a akvizic. Z tohoto důvodu mají firmy následující výhody:

- Žádné spojené licenční poplatky za PostgreSQL.
- Neomezený počet nasazení PostgreSQL.
- Ziskovější obchodní model.

3.4.7.2 Výhody pro uživatele

PostgreSQL je velmi atraktivní pro vývojáře, administrátory a architekty. Nabízí bohaté funkce, které umožňují vývojářům pracovat agilně. Následují některé z funkcí, které jsou pro vývojáře přitažlivé (Juba, 2019):

- Nová verze je vydávána téměř každý rok; dosud bylo vydáno 25 hlavních verzí, počínaje PostgreSQL 6.0.
- Velmi dobrá dokumentace a aktivní komunita umožňuje vývojářům rychle nalézt řešení problémů. Manuál PostgreSQL má více než 2,500 stránek.
- Bohatý repositář rozšíření umožňuje vývojářům soustředit se na obchodní logiku. Také usnadňuje adaptaci na změny požadavků.
- Zdrojový kód je k dispozici zdarma. Lze jej upravovat a rozšiřovat bez velké námahy. Existuje mnoho nástrojů a utilit pro PostgreSQL.
- Bohaté klientské a správcovské nástroje umožňují vývojářům provádět rutinní úkoly, jako je popis databázových objektů, export a import dat a rychlé vytváření záloh databází.
- Úkoly správy databáze nevyžadují mnoho času a mohou být automatizovány.
- PostgreSQL lze snadno integrovat s dalšími systémy pro správu databází, čímž poskytuje softwarové architektuře dobrou flexibilitu pro implementaci softwarových návrhů.

3.4.8 KeyCloak

KeyCloak je správa identit a přístupu s otevřeným zdrojovým kódem. Tato služba přidává ověřování do aplikací a zabezpečuje služby s minimálním úsilím. Tudíž není třeba řešit správu ukládání uživatelů do databází a jejich následné ověřování. (Keycloak Authors, 2023)

KeyCloak přináší několik výhod a to jsou:

Jednotné přihlašování

Uživatelé se ověřují pomocí Keycloak namísto jednotlivých aplikací. To znamená, že aplikace se nemusí zabývat přihlašovacími formuláři, ověřováním uživatelů a ukládáním uživatelů. Jakmile se uživatel přihlásí do Keycloak, nemusí se znovu přihlašovat při přístupu k jiné aplikaci. (Keycloak Authors, 2023)

To platí i pro odhlášení. Keycloak poskytuje jednotné odhlášení, což znamená, že uživatelé se musí odhlásit pouze jednou, aby byli odhlášeni ze všech aplikací, které Keycloak používají. (Keycloak Authors, 2023)

Propojení Identit a Sociální Přihlášení

Povolení přihlášení pomocí sociálních sítí je snadné. Lze přidat prostřednictvím administrační konzole. Stačí vybrat sociální síť, kterou je potřeba přidat a není k tomu zapotřebí psát žádný kód ani provádět změny v konkrétní aplikaci. (Keycloak Authors, 2023)

Keycloak také může ověřovat uživatele s existujícími poskytovateli identit OpenID Connect nebo SAML 2.0. Opět je to jen otázka konfigurace poskytovatele identit prostřednictvím administrační konzole. (Keycloak Authors, 2023)

Autorizační služby

Pokud role založená autorizace nevyhovuje specifikovaným potřebám, Keycloak poskytuje také detailní autorizační služby. To umožňuje spravovat oprávnění pro všechny služby z administrační konzole Keycloak a dává příležitost přesně definovat politiky, které jsou potřeba. (Keycloak Authors, 2023)

Uživatelské federace

Keycloak má vestavěnou podporu pro připojení k existujícím serverům LDAP nebo Active Directory. Pokud jsou uživatelé v jiných úložištích, například v relační databázi je možnost také implementovat vlastního zprostředkovatele. (Keycloak Authors, 2023)

3.4.9 NPM

Npm je největší registr softwaru na světě. Vývojáři open source ze všech kontinentů používají npm ke sdílení a půjčování balíčků a mnoho organizací používá npm také ke správě soukromého vývoje. (Karrys, 2023)

Dle dokumentace (Karrys, 2023) se npm se skládá ze tří různých částí a to jsou:

- Webové stránky – všechny seznamy balíčků

- Rozhraní příkazového řádku – CLI
- Samotný registr

Webové stránky slouží k vyhledávání balíčků, nastavování profilů a správě dalších aspektů práce s npm. Můžete například nastavit organizace pro správu přístupu k veřejným nebo soukromým balíčkům.Rozhraní CLI se spouští z terminálu a je to způsob, jakým většina vývojářů komunikuje s npm.Registr je rozsáhlá veřejná databáze softwaru JavaScript a metainformací o něm. (Karrys, 2023)

Dle dokumentace (Karrys, 2023) npm mohou vývojáři využívat různě:

- Přizpůsobit balíčky kódu pro své aplikace nebo začlenit balíčky tak, jak jsou.
- Stáhnout samostatné nástroje, které jsou ihned k použití.
- Spouštět balíčky bez stahování pomocí npx.
- Sdílet kód s libovolným uživatelem npm.
- Omezit kód na konkrétní vývojáře.
- Vytvářet organizace pro koordinaci údržby balíčků, kódování a vývojářů.
- Vytvářet virtuální týmy pomocí organizací.
- Spravovat více verzí a závislostí kódu.
- Provádět snadnou aktualizaci aplikací při aktualizaci základního kódu.

3.4.10 Vue.js

Vue.js je moderní progresivní JavaScriptový framework pro frontendový vývoj aplikací. Staví na standartních jazycích HTML, CSS a JavaScript. Poskytuje deklarativní a na komponentách založený přístup pro tvorbu uživatelských rozhraní od malého měřítka až po složité rozhraní. (You, c2014–2023)

Vue.js pokrává většinu funkcí, které jsou potřebné pro vývoj frontendu. Jeho hlavní výhodou je snadná osvojitelnost, jelikož web jako takový je pojem, který může nabývat jednoduchosti, ale lze také řešit složité rozsahy a funkčnost, proto je Vue.js navrženo tak, aby se dalo využít v širokém spektru použitelnosti. Nabízí možnost vylepšení statického HTML kódu bez nutnosti sestavení, tvorbu HTML komponent a jejich vložení do stránky, což vede ke zlepšení kvalitu kódu a jeho znuvupoužitelnosti. V posledních letech se rozjel trend SPA aplikací (single-page application), což Vue také nabízí. Některá uživatelská rozhraní jsou tvořena pro běžné aplikace, nejedná se tedy o interní aplikace, takže zde jsou kladeny nároky na kvalitní SEO a optimalizaci HTML kódu, což zajišťuje SSR (server-side

rendering), které vykreslí kompletní HTML kód pro potřeby vyhledávacích robotů. (You, c2014–2023)

3.4.10.1 Přístupy ke psaní kódu

Vue.js nabízí dvě možnosti přístupu k psaní kódu. Ve verzi 2 nabízelo pouze jednu, a to konkrétně optionAPI. V tomto přístupu jsou jasně definované metody, watchery, životní cykly aplikace, propsy a konstanty. (You, c2014–2023)

Nevýhody:

- Organizace kódu – kód je shlukován podle typu, nikoli podle logiky
- Znovupoužitelnost – težší sdílení logiky mezi komponentami
- Typescript – horčí integrace s typescriptem
- Reaktivita – nutnost rozlišovat co je a co není deklarováno v datech
- Abstrakce – složitější logika může být hůře implementovatelná
- Optimalizace pro budoucí vývoj – přechod na Vue 3 a compositionAPI

Obrázek 11 Rozdíl mezi Option a Composition API (You, c2014–2023)

S verzí 3 přišel nový přístup, composition API, který má inspiraci v Reactu. Na rozdíl od optionAPI se zde využívá importování funkcí a přináší několik nových prvků, jako například reaktivitu – ref(), reactive(), přístup k životnímu cyklu aplikace – onMounted() a

dependency injection, které umožňuje využít systém Vue pro injectování závislotí z rozhraní API Reactivity (You, 2023). Je zde také lepší logika uspořádání kódu, kde jsou u sebe jednotlivé části, které spolu souvisí a nejsou roztroušené po celém souboru. Další novinkou je využití **setup** metody, která je kompilována jako inline funkce a šablona má k deklarovaným proměnným přímí přístup a není nutné vytvářet proxy instanci k jejich dosažení. (You, c2014–2023)

3.4.11 Bootstrap 5

Bootstrap je sada nástrojů určená pro vývoj frontednových řešení pomocí HTML, CSS a JavaScript. Jeho princip je založený na hotových komponentách, které se mohou vkládat libovolně do kódu a dle potřeby upravovat.

3.4.11.1 Instalace

Instalace Bootstrapu je možná 2 způsoby. Prvním z nich je importovat knihovny .css a .js pomocí CDN linků. Tato verze má však vždy nejaktuálnější verzi, proto je třeba kontrola, zda je kód kompatibilní s danou verzí či nikoli. Její výhodou je rychlý import knihovny do stránky. (Bootstrap Team, [2023])

```
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.r
```

Obrázek 12 CDN linky pro použití Bootstrapu (Bootstrap Team, [2023])

Druhým způsobem dle dokumentace (Bootstrap Team, [2023]) je stažení knihovny lokálně. Zde je navýběr z několika možností. Lze stáhnout:

- kompilované soubory .js a .css
- zdrojové soubory
    - SASS kompilátor – kompiluje Sass zdrojové kódy do CSS kódu
    - autoprefixer pro prefixy CSS z vendor balíčků.
- manažer závislostí – zde Bootstrap nabízí npm, yarn, RubyGems, composer, NuGet

```
$ npm install bootstrap@5.3.0-alpha1
```

Obrázek 13 Instalační skript npm pro Bootstrap (Bootstrap Team, [2023])

Tento způsob nabízí vetší kontrolu nad celkovou funkčností kódu. Balíček Bootstrapu je stažen lokálně do složky projektu a veškeré potřebné komponenty jsou vykreslovány z lokální vendor složky. Je zde tedy kontrola nad verzí Bootstrapu, při využití preprocesorů i nad jejich verzí, verzí loaderů a projekt si tak zachovává svou kontinuitu po dobu, kdy je projekt ručně aktualizován.

3.4.11.2 Přizpůsobení importu CSS stylů

Pokud je knihovna nainstalována lokálně, a v projektu je využitý Sass preprocesor pro css, Bootstrap nabízí možnost přizpůsobení importu obsahu. Lze importovat celou Sass strukturu, která obsahuje veškeré komponenty, které Bootstrap nabízí. To ovšem vede také ke zvětšení velikosti projektu po jeho sestavení do produkce, proto je zde možnost importu jen některých částí knihovny. Například je možné v hlavním souboru .scss importovat pouze grid systém. Následně bude základní soubor .css obsahovat pouze kód, který se týká grid systému a nebude obsahovat žádný další kód, který by byl nevyužitý. Tímto způsobem se tedy dá dosáhnout pěkné optimalizace kódu, jeho znuvupoužitelnosti v celém rozsahu aplikace a ke zmenšení buildu aplikace. (Bootstrap Team, [2023])

Bootstrap využívá předem definované hodnoty pro odsazení, margin, padding, výpočty velikosti sloupců a dalších atributů. I tyto hodnoty lze jednoduše upravit při importu určitého souboru a lze tak změnit předem definované hodnoty.

3.4.12 Axios

Axios je HTTP klient pro node.js a prohlížeče založený na promise-based. Je izomorfní (může běžet v prohlížeči i nodejs se stejnou kódovou základnou). Na straně serveru používá nativní modul node.js HTTP, zatímco na straně klienta (prohlížeče) používá XMLHttpRequests. (Getting Started, [2022])
Axios se do projektu importuje jednoduše, a to následujícím způsobem:

```
import axios from 'axios';
```

Obrázek 14 Import balíčku do projektu (vlastní zdroj)

Základní metody jsou **GET**, **POST**, **PUT**, **PATCH**, **DELETE**. Následující funkce popisuje získaní dat ze vzdáleného serveru, kdy je použita metoda GET, kde je možno

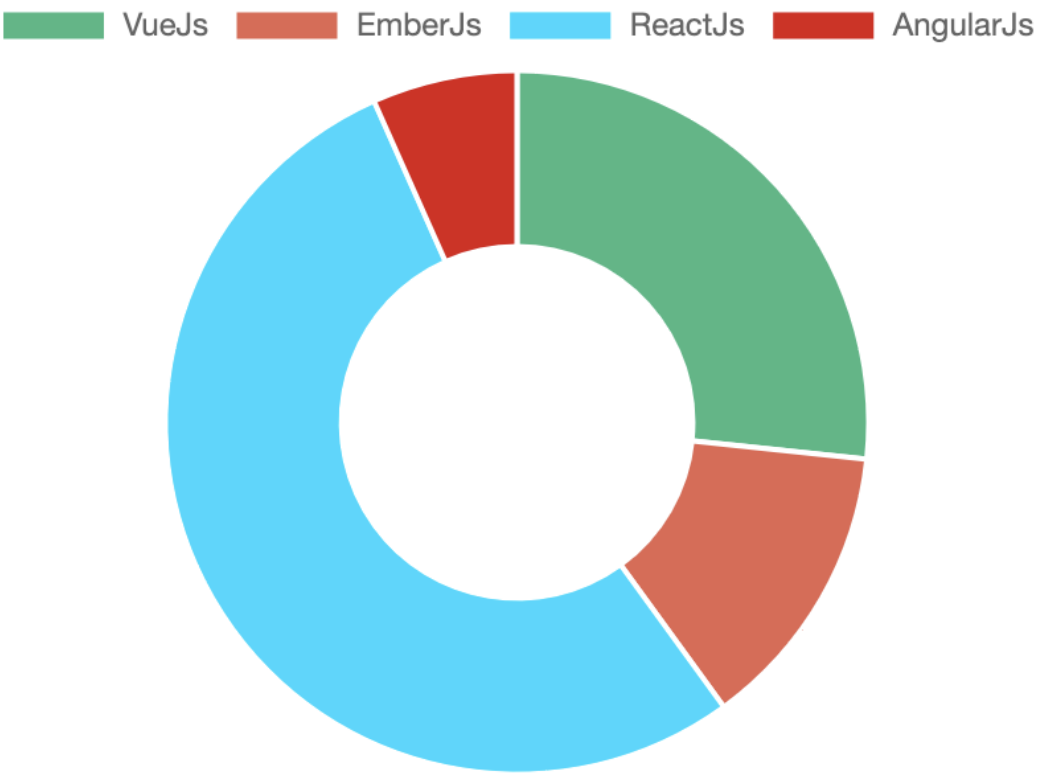
specifikovat další nastavení pro hlavičky. Dále se zprostředkují data a pokud dojde k chybám v datech, je zachycen error, který detailně popisuje chybu.

```
axios.get('https://example.com/api/data', {
  headers: {
    Authorization: 'Bearer my-token'
  }
})
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error(error);
  });
```

Obrázek 15 Metoda get použitím Axios balíčku (vlastní zdroj)

3.4.13 Vue Charts

Vue charts je obalovací knihovna pro Chart.js. Tato knihovna přináší snadnou implementaci grafů do Vue.js a jejich snadné využití skrze komponenty projektu. Hlavní výhoda knihovny je ta, že abstrahuje základní logiku core balíčku a přináší tak rychlost a velkou flexibilitu.



Obrázek 16 Příklad grafu pomocí Vue charts (Vue-charts team, [2016])

Instalace je jednoduchá. Stačí v projektu spustit npm skript, který nainstaluje danou knihovnu. Jediné, co je potřeba udělat navíc, je přidat do závislostí i Chart.js, jelikož Vue charts je závislá na této vrstvě. (Vue-charts team, [2016])

Tato knihovna nabízí velkou škálu grafů. Koláčové grafy, histogramy, lineární grafy a spoustu dalších. Velkou výhodou je, že všechny data v těchto grafech mohou být reaktivní, a tudíž se mohou měnit za běhu aplikace. Další velkou výhodou je kombinování grafů a vytvoření vlastního grafu pro specifické účely. (Vue-charts team, [2016])

3.4.14 SASS

SASS je preprocesor skriptovacího jazyka, který byl navržen pro rozšíření funkcí základního jazyka CSS. Umožňuje vývojářům psát kód efektivněji a s většími možnostmi než standardní CSS.

SASS byl poprvé představen v roce 2006 a byl vytvořen Hamptonem Catlinem a Natalie Weizenbaum. K jeho vzniku vedlo to, že CSS nebylo na takové úrovni, jako je dnes a vývojářům chyběla absence některých důležitých funkcí, jako třeba dědičnost, mixiny a struktura samotného. Byl tedy vyvinut jako reakce na potřebu lepší strukturovanosti a funkcí v CSS.

Klíčové vlastnosti a výhody (The Sass team, c2006–2023)**:**

- **Proměnné:** Umožňují definování hodnot, které mohou být opakovaně použity po celém stylu.
- **Vnořené pravidla:** Umožňují definovat styly uvnitř jiných stylů, což zlepšuje čitelnost a snižuje opakování kódu.
- **Mixiny:** Umožňují vytvářet znovupoužitelné bloky stylů, které lze vložit do různých částí SASS souboru.
- **Funkce a operace:** Pro dynamické výpočty s hodnotami (např. barvy, velikosti atd.).
- **Dědičnost:** Umožňuje sdílet soubor pravidel mezi různými selektory.
- **Importy a moduly:** Umožňují rozdělit kód do menších souborů a následně je kombinovat do jednoho kompilovaného CSS souboru.

SASS s sebou nese ale i jednu nevýhodu, kterou je nutnost kompilace kódu do čistého jazyka CSS. Pokud bz nebyl kód kompilován, běžné prohlížeče by si s ním nedokázali poradit a styly by tudíž nebyli načteny. V dnešní době jsou již vyhledávače na vysoké úrovni a dokážou si poradit i s neúplným HTML kódem, který si sami při vykreslování doplní, ale

s kompilací kódu psaného s využitím preprocesorů si poradit nedokážou. (The Sass team, c2006–2023)

3.5 Časové řady

Data časových řad, jsou posloupností datových bodů indexovaných v časovém pořadí. Tyto datové body se obvykle skládají z po sobě jdoucích měření provedených ze stejného zdroje v pevně stanoveném časovém intervalu a používají se ke sledování změn v čase. (InfluxData Inc., c2023)

Časová řada dat je soubor pozorování získaných opakovaným měřením v průběhu času. Pokud tyto body zakreslíte do grafu, bude jednou z os vždy čas. Metriky časových řad se vztahují k části dat, která se sledují s určitým časovým přírůstkem. Metrika by se například mohla týkat toho, kolik zásob se prodalo v obchodě od jednoho dne do druhého. Data časových řad jsou všude, protože čas je součástí všeho, co je pozorovatelné. Jak je náš svět stále více vybaven přístroji, senzory a systémy neustále vysílají neúnavný proud dat časových řad. Taková data mají četné využití v různých odvětvích. (InfluxData Inc., c2023)

Příklady dle zdroje (InfluxData Inc., c2023) analýzy časových řad:

- Elektrická aktivita v mozku
- Měření dešťových srážek
- Ceny akcií
- Počet slunečních skvrn
- Roční maloobchodní tržby
- Měsíční počet předplatitelů
- Počet úderů srdce za minutu

Graf časové řady

Grafy časových řad jsou jednoduše grafy časových řad dat na jedné ose (obvykle Y) proti času na druhé ose (obvykle X). Grafy bodů časové řady dat mohou často ilustrovat trendy nebo vzorce přístupnějším a intuitivnějším způsobem. (InfluxData Inc., c2023)

Statistiky časových řad

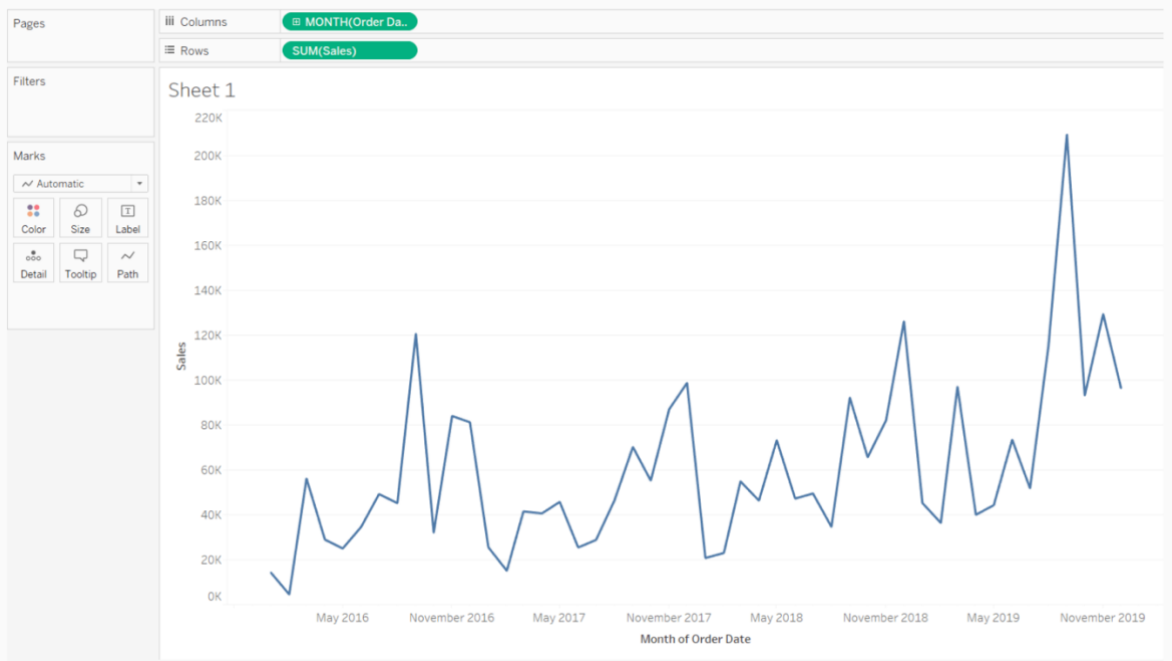
Graf časové řady je graf, v němž osa x představuje určitou časovou míru. Ve skutečnosti je osa x označena jako časová osa. Osa y představuje měřenou proměnnou. Datové body jsou ve většině případů zobrazeny a spojeny přímkami, což umožňuje interpretaci výsledného grafu. (InfluxData Inc., c2023)

3.5.1 Analýza časových řad

Analýza časových řad je specifický způsob analýzy posloupnosti datových bodů shromážděných za určitý časový interval. Při analýze časových řad analytici zaznamenávají datové body v konzistentních intervalech po určitou dobu, nikoliv pouze přerušovaně nebo náhodně. Tento typ analýzy však není pouhým shromažďováním dat v průběhu času. (Tableau Software, c2003–2023)

To, co odlišuje údaje v časových řadách od jiných údajů, je, že analýza může ukázat, jak se proměnné v čase mění. Jinými slovy, čas je klíčovou proměnnou, protože ukazuje, jak se data v průběhu datových bodů upravují, stejně jako konečné výsledky. Poskytuje další zdroj informací a stanovené pořadí závislostí mezi daty. (Tableau Software, c2003–2023)

Analýza časových řad obvykle vyžaduje velký počet datových bodů, aby byla zajištěna konzistence a spolehlivost. Rozsáhlý soubor dat zajišťuje, že je k dispozici reprezentativní velikost vzorku a že analýza dokáže projít daty. Zajišťuje také, že případné zjištěné trendy nebo vzorce nejsou odlehlé a že lze zohlednit sezónní odchylky. Kromě toho lze data časových řad použít k prognózování – předpovídání budoucích údajů na základě historických dat. (Tableau Software, c2003–2023)



Obrázek 17 Příklad analýzy časové řady (Tableau Software, c2003–2023)

4 Vlastní práce

4.1 Analýza současného stavu

Současný stav systému, který řeší prodej produktů ve firmě – jedná se o zatím o několik druhů limonád, je následující. V lednicích v jednotlivých patrech jsou k dispozici všem zaměstnancům limonády. Každý, kdo si chce vzít limonádu, se musí napsat na tabuli u lednice, tedy dělat si čárky pro jednotlivé druhy limonád. Každý měsíc se tato tabule vyfotí a pošle na BackOffice, kde ji zaměstnankyně přepíšou do Excelu. V Excelu je typicky název limonády, její cena a pak zaměstnanci, kteří si danou limonádu vzali. U zaměstnance je tedy počet kusů určité limonády, kolik má zaplatit a kolik dluží z minula. U každé tabulky je pak QR kód, kam se má platba odeslat.

Tento celý proces je velmi zdlouhavý a může docházet k chybám při přepisování anebo přímo při zapisování konkrétního počtu zaměstnancem. Také není jasné, kolik limonád ještě zbývá v jednotlivých lokalitách a jestli jsou vůbec ještě nějaké k dispozici pro doplnění. Další věcí je to, že vždy musí někdo, kdo má limonády na starost dát vědět do společné komunikace, že je připraveno vyúčtování, což s sebou nese následky, že ně každý si zprávu přečte a snadno se tak stane, že zjistí po pár měsících, že dlužná částka je poněkud vyšší.

4.1.1 SWOT analýza současného stavu

Silné stránky (Strengths)

- Jednoduchý systém, který nevyžaduje složité technologie.
- Přímá kontrola spotřeby limonád u každé lednice.
- Možnost vizuální kontroly záznamů pro všechny zaměstnance u lednice.
- QR kód pro snadný a rychlý způsob platby.

Slabé stránky (Weaknesses)

- Velký potenciál chyb při manuálním záznamu a přepisování dat.
- Zdlouhavý proces sběru, zpracování a vyúčtování informací.
- Neschopnost monitorovat aktuální zásoby limonád v reálném čase.
- Závislost na časté komunikaci a upozorněních mezi zaměstnanci.
- Nehomogenní systém – každá lednice může mít jiný systém záznamu.
- Nedostatek přehledu o dlouhodobých dlužnících.

Příležitosti (Opportunities)

- Implementace digitálního systému pro sledování a vyúčtování.
- Možnost zavedení automatických upozornění pro dlužníky.
- Vytvoření systému pro monitorování zásob v reálném čase.
- Vývoj mobilní aplikace nebo webového rozhraní pro zaměstnance pro snazší přístup a záznamy.
- Integrace s platovými systémy pro automatické vyúčtování.

Hrozby (Threats)

- Odpor zaměstnanců k novému řešení.
- Finanční náklady na vývoj a implementaci nového systému.
- Potenciál pro technické problémy nebo poruchy v novém systému.
- Zvýšená závislost na technologii může způsobit problémy v případě jejího selhání.

4.2 Analýza požadavků pro vytvoření nového systému

Na základě rozhovorů se zaměstnanci BackOffice, kteří budou obsluhovat interní systém, kolegy z firmy a studiu dostupných, již existujících řešení byly formulovány požadavky a cíl, který by měl informační systém naplnit.
Z těchto rozhovorů vzniklo následující:

- **Cíl systému**
    - Optimalizace procesů obsluhy doplnění a prodeje nápojů
    - Ušetření času při měsíčním vyúčtování prodaných produktů
    - Optimalizace vyúčtování
    - Zlepšení přehledu o produktech na jednotlivých lokalitách a centrálním skladu
- **Uživatelské role a oprávnění**
    1. Administrátor
        - obsluha doplňování zboží na sklad,
        - distribuce na jednotlivé lokality
        - Změna statusu u plateb
    2. Uživatel
        - Skenování lokality

    - Vybrání produktu
    - Sledování financí – kolik má zaplaceno, kolik ještě dluží
- **Funkční požadavky**
    1. Administrace
        - Nástěnka – přehled produktů, analytické nástroje, stav produktů
        - Základní sklad
        - Lokality
        - Generování QR kódů
        - Stav plateb
        - Posílání notifikací o zaplacení
    2. Uživatelská část
        - Skenování lokality
        - Vytvoření objednávky
        - Správa financí
        - Manažer objednávek
        - Historie účtu
- **Nefunkční požadavky**
    - Bezpečnost – šifrované připojení
    - Intuitivnost – jednoduché a přehledné uživatelské rozhraní
    - Výkon – rychlá odezva systému, minimalizování blokací
- **Technické požadavky**
    - Využití vnitropodnikového autorizačního adaptéru
    - Kompatibilita s prohlížeči
    - Mobilní přístup
    - API – vystavení BE aplikace, zabezpečené
- **Budoucí rozšíření**
    - Možnost vytvoření mobilní aplikace
    - Integrace s interním systémem financí
    - Možnost přidání modulu pro správu zdrojů
    - Možnost integrace platební brány
    - Uživatelská peněženka – možnost nahrání peněz pro budoucí platby
    - Propojení HW lednice a aplikace – otevření po naskenování lokality

- **Omezení**
    - Omezený rozpočet
    - Přesný vývoj na zakázku pro využití interních potřeb
    - Údržba – aby byl schopný systém kdokoli rozšířit a upravovat
    - Interní architektura serveru a prostředí

Nový systém, který budu vytvářet by měl velmi usnadnit práci se zásobami, ušetřit provozní náklady, jelikož zaměstnanci, kteří s ním budou pracovat nebudou trávit spoustu času přepisováním informací, což se jedná o ušetření hodin týdně. Další výhodou bude konzistentnost dat, jelikož data v systému budou ucelena a v jedné podobě, tudíž nad nimi půjdou dělat různé operace pro statistiky týkající se produktů a implementace jednoduchého skladového systému dle potřeb pomůže zpřehlednit jejich stav a potřebu distribuce do jednotlivých lokalit. Zaměstnanci využívající tento systém budou mít jasný přehled o svých financích, kolik zaplatili a kolik mají zaplatit a administrátoři jasný přehled o všech objednávkách a o tom, kdo již zaplatil a kdo ne.

4.3 Návrh systému

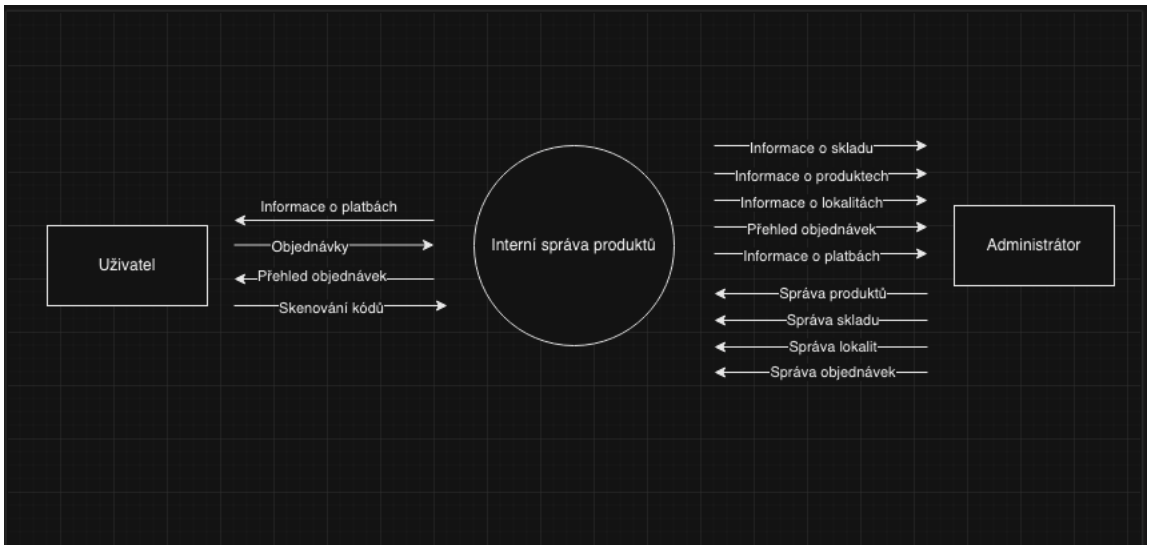
Základní kamenem celého systému je databáze. Zde jsem zvolil na základě zkušeností a předchozí práce PostgresSQL. Pro komunikaci a zpracování veškerých dat jsem zvolil php Framework Symfony, který je robustní a ve firmě rozšířený, tudíž je do jisté míry připravené prostředí pro testování a následné nasazení do produkce. Pro vývoj a následný běh aplikace bez problému a zaručení stejného chování na všech prostředích jsem celou aplikaci dockerizoval, tudíž poběží stejně ve všech prostředích a stejně tak, kdyby někdo jiný pokračoval ve vývoji, zaručí se tím stabilita a integrita celé aplikace.

Zvolil jsem architekturu vystavení API, jelikož je to běžný přístup a má několik výhod oproti monolitním aplikacím. Jako velkou výhodu vidím v tom, že backend aplikace je odstíněný oproti frontendové části, tudíž komunikace probíhá výhradně přes API a na toto API je možné připojit jakoukoli aplikaci, která umožňuje komunikaci se serverem, například mobilní aplikace. Pro frontendovou část, jsem vybral Vue.js, javascriptový Framework, který jsem zvolil proto, že v něm pár let pracuji, je rychlý a má všechny potřebné funkce pro vytvoření přehledného, funkčního a zabezpečeného uživatelského rozhraní, veškeré nástroje pro vytvoření zabezpečené aplikace.

Zabezpečení aplikace bude zajišťovat služba KeyCloak, která je propojena s vnitropodnikovým systémem, který spravuje všechny zaměstnance ve firmě, takže nebude docházet ke komplikacím při jejich úpravě nebo deaktivování přistupu.

4.3.1 DFD diagram

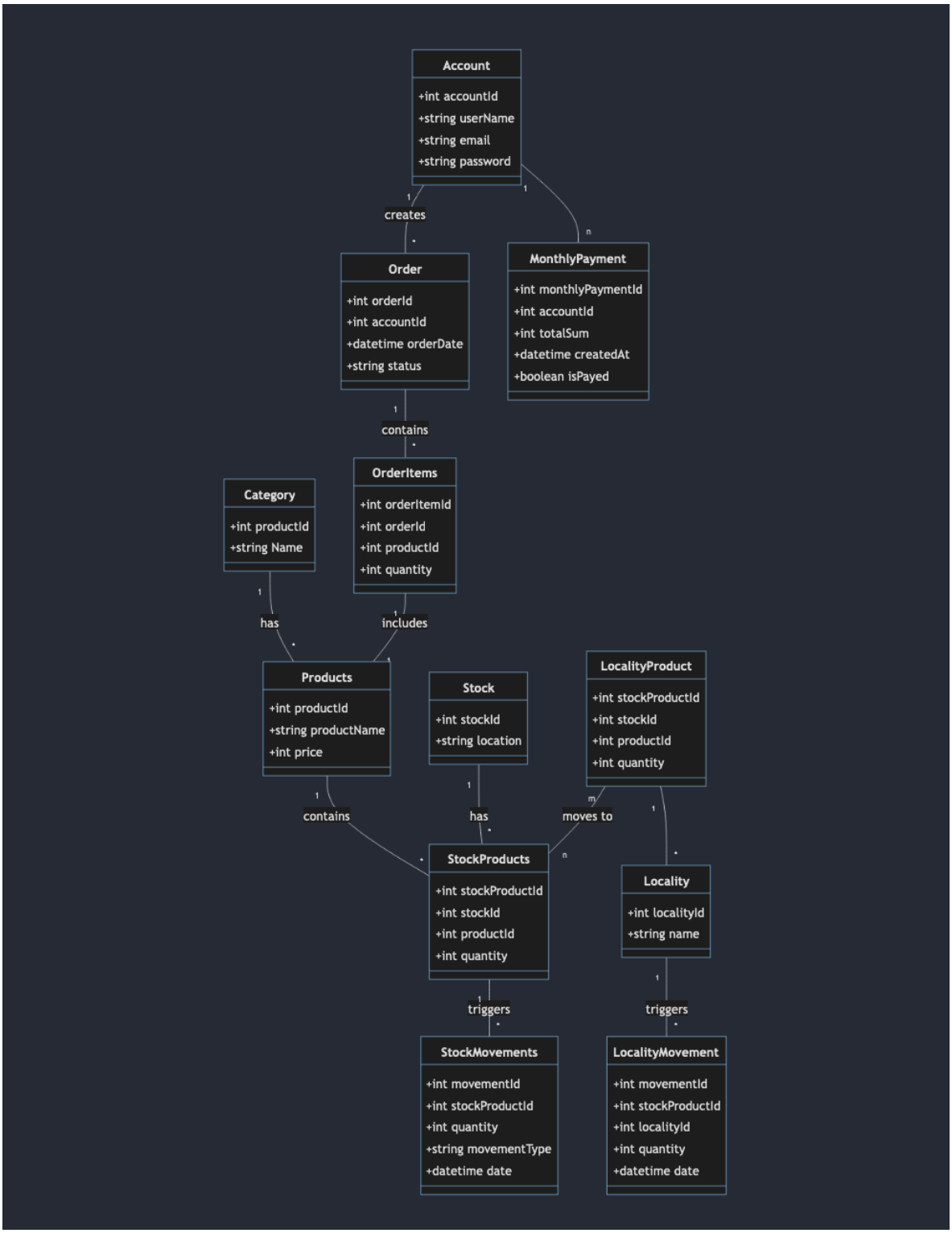
Na zobrazeném digramu popisuji základní toky, které jdou buď do systému nebo ze systému. Dvě externí entity – uživatel a administrátor komunikují že systémem. Zde bych chtěl zdůraznit, že administrátor má stejná práva jako uživatel. Uživatel může skenovat kódy na jednotlivých lokalitách, může pak vytvořit objednávku a ze systému si může vyžádat svůj přehled objednávek a celkový přehled o zaplacených a nezaplacených objednávkách.
Administrátor si ze systému může vyžádat přehledy o lokalitách, o konkrétních skladech, o produktech a o všech objednávkách a jejich stavu. Administrátor může dále upravovat stav jednotlivých produktů, skladů a lokalit. Dále také může spravovat jednotlivé objednávky a měnit jejich stav, zda jsou zaplacené či nikoli.



Obrázek 18 DFD diagram systému level 0 (vlastní zdroj)

4.4 Backendová část

4.4.1 Class diagram



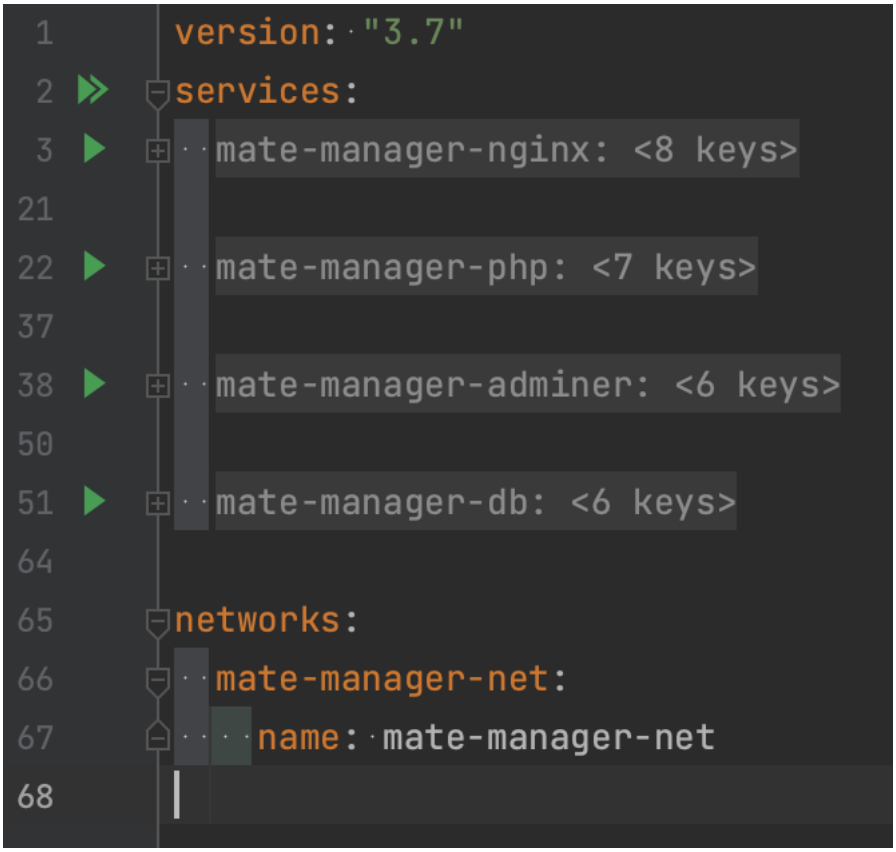
Obrázek 19 Class diagram systému (vlastní zdroj)

4.4.2 Základní konfigurace vývojového prostředí

Pro prvotní přípravu aplikace bylo nutné nejdříve napsat vývojové prostředí. Pro tento účel jsem zvolil Docker, jelikož je to ta nejlepší možnost, při které dostanu záruku toho, že aplikace poběží na všech prostředích úplně stejně a bude se i stejně chovat.
Základní konfigurace se skládá z těchto částí:

- nginx
- php-fpm
- .env
- docker-compose.yaml

Nginx jsem nakonfiguroval podle dokumentace a potřeby aplikace. Vytvořil jsem development konfiguraci, kde jsem specifikoval vlastnosti serveru, na jaké adrese poběží, co na něm vůbec poběží, kromě aplikace samotné na ně, ještě běží GUI pro databázi adminer. Image nginx jsem napsal do Dockerfilu, který si stahne image z firemní Artifactory a dále s ním bude pracovat.
Stejně podobným způsobem jsem implementoval i php-fpm. Vytvořením konfiguračních souborů jsem si zvolil to, co chci, aby implementace php obsahovala. V php.ini souboru jsem specifikoval cesty k logům, timezone, a ostatní základní nastavení php. Většina těchto souborů je psaná podle dokumentace. (odkaz?)Dockerfile si také sahá na firemní artifactory pro php 8.1 a následně jsem definoval instalaci opcache nebo instalaci pgsql driveru pro postgres databázi. Jako poslední se zkopírují konfigurace, vytvoří se php-fpm složka a volume app, která se nastaví na workdir, kterou budu využívat v další konfiguraci.
V .env souboru jsem specifikoval pouze název projektu pro lepší přehlednost.



Obrázek 20 Služby vytvořené v Dockeru (vlastní zdroj)

Jako poslední jsem definoval soubor docker-compose.yaml. V tomto souboru jsem vytvořil jednotlivé služby, tedy nginx, adminer, postgres a php. Aplikace tedy běží na 4 kontejnerech. Na obrázku nahoře je vidět struktura souboru. Ještě jsem vytvořil network, které mají jednotlivé služby přiřazeny, v rámci něho mají svůj alias pro to, aby byly dosažitelné i mimo kontejner, například při specifikaci připojení k databázi.

4.4.3 Inicializace symfony

Symfony jsem vytvořil pomocí symfony cli skriptu. Jako první jsem zvolil aktuální stable verzi symfony 6.3.*. Následně je nutné definovat, s jakou databází budu pracovat v rámci aplikace. Zvolil jsem PostgreSQL, což je objektová databáze. Definici užití této databáze jsem definoval následujícím způsobem.

DATABASE_URL="postgresql://dbuser:dbpasswd@dbhost:5432/dbname"

Tím jsem aplikaci řekl, že na této adresu, s určitým jménem a heslem k databázi, na určitém hostu se specifikovaným portem chci vytvořit databázi se specifickým jménem. Následně musím říct aplikaci, že chci danou databázi podle konfigurace vytvořit pomocí příkazu, který vypadá následovně *bin/console doctrine:database:create*. Tímto si vytvořím prázdnou databázi, se kterou můžu pracovat.

Jako další důležitou věc jsem musel vyřešit otázku CORS. Tedy to, abych mohl komunikovat s API, musím přidat parametr Access-Control-Allow-Origin do hlavičky odpovědi, aby mi je můj prohlížeč neblokoval a já jsem taky mohl dostat v pořádku odpověď. Pro tento účel slouží balíček nelmio/cors-bundle, který se stará právě o to a dokonale funguje s další věcí, kterou jsem do projektu zahrnul, a to je API Platform.

Api Platform jsem si zvolil na základě osobních zkušeností, výborné dokumentace a ulehčení od tvoření několika controllerů pro jednotlivé tabulky znovu a znovu. Konfigurace byla jednoduchá. Po instalaci mi symfony vygenerovala konfigurační soubor, ve kterém jsem nastavil vše, co budu potřebovat, jako například formát pro zobrazovaní dat a komunikaci, a jako formát jsem zvolil obyčejný application/json, byť jsem měl na výběr z několika formátů, ale pro tento účel bude stačit tento. Dále jsem nastavil swagger pro dokumentaci API a pro pozdější testování a jako poslední jsem specifikoval chybové formáty a patch formáty, oba, na stejný, tedy api+json.

```yaml
# api/config/packages/api_platform.yaml
api_platform:
    formats:
        json: ['application/json']
    swagger:
        api_keys:
            access-token:
                name: Authorization
                type: header
    patch_formats:
        json: [ 'application/merge-patch+json' ]
        jsonapi: [ 'application/vnd.api+json' ]
    error_formats:
        jsonproblem: [ 'application/problem+json' ]
        jsonapi: [ 'application/vnd.api+json' ]
```

Obrázek 21 Konfigurace Api platfrom pro projekt (vlastní zdroj)

Tímto jsem zakončil základní konfiguraci aplikace. Pro ulehčení práce s dockerem jsem ještě vytvořil Makefile, ve kterém jsem specifikoval několik nejpoužívanějších příkazů, které pak budu moci spouštět pomocí make *name*v rootu projektu. Jedná se především o php

příkazy, které musím spouštět přímo uvnitř kontejneru a pak o příkazy spojené s vytvořením kontejnerů, jejich restartem nebo spuštěním.

4.4.4 Autentizace uživatele

V tomto projektu nebudu využívat klasickou autentizaci, tedy že se nebudou data z loginu odesílat na api a to následně ověří existujícího uživatele a jeho práva, vytvoří token a vrátí ho zpět k dalšímu užití. Jelikož využívám keycloak na frontendu, token již bude vygenerovaný a já na to budu muset reagovat. Když přijde požadavek, který má v hlavičce Authorization parametr, který obsahuje token, tak si vzít daný token, dekódovat ho. V tomto případě budu do databáze ukládat pouze jen některé informace, jelikož pokud přijde daný token, je jasné, že tento uživatel má přístup a ověřil se již na frontendu aplikace.

Jako první jsem vytvořil službu KeycloakAuthenticator. Tato služba implementuje AbstractAuthenticator, tedy její metody supports, authenticate a pak rekaci na úspěšný a neúspěšný proces autentizace. Jádro služby spočívá práve v metodě authenticate. V této metodě si z hlavičky requestu vezmu parametr Authorization a jelikož její hodnota vždy přichází ve tvaru ‘Bearer <token>’, musím odstranit pomocí regexu slovo Bearer a mezeru, abych získla pouze token. Jedná se JWT token, tedy token, který je složený ze tří částí. Zde

```
1 usage   Pavel Jakl *
104     private function getKey(): array
105     {
106         $jwkData = $this->cacheApp->get('jwk_keys', function(ItemInterface $item) {
107             $jwkData = json_decode(
108                 file_get_contents(sprintf(
109                     format: '%s/realms/%s/protocol/openid-connect/certs',
110                     trim($this->parameterBag->get('keycloak_url'), characters: '/'),
111                     $this->parameterBag->get('keycloak_realm'),
112                 )),
113                 associative: true
114             );
115
116             $item->expiresAfter( time: 3600);
117             $item->tag(['authentication']);
118
119             return $jwkData;
120         });
121
122         return JWK::parseKeySet($jwkData);
123     }
124 }
```

Obrázek 22 Služba pro získání autentizačního klíče (vlastní zdroj)

je validace, která mi řekne, zda je token validní, tedy se skládá právě ze třech částí. Pokud ne, je vyhozena výjimka na nevalidní token. Pokud je vše v pořádku, dekóduji token token. Zde je důležité definovat klíč a šifrování, aby byl token správně definován.

Klíč k šifrování získám pomocnou funkcí, která si vezme obsah stránky s definovanou url adresou a realmem ke keycloaku, nastaví expiraci šifrování a vezme si specifickou hodnotu. Následně vrátí set klíčů, který byl použit k šifrování tokenu.

Algoritmus pro šifrování získám z tokenu, respektive jeho první části. Pokud je vše správně, token můžu dekódovat. Když je token dekódovaný, vezmu si ještě jednu informaci, a to, zda je uživatel podle interního systému v roli administrátora, či pouze uživatele. Pokud se to nepovede, aplikace vyhodí chybu, proč nešlo token dekódovat.

```
Pavel Jakl *
public function authenticate(Request $request): Passport
{
    $JWTtoken = $request->headers->get( key: 'Authorization');
    preg_match( pattern: '/Bearer\s(\S+)/', $JWTtoken, &: $matches);
    $a = $matches[1];
    $parts = explode( separator: '.', $a);
    if (count($parts) !== 3) {
        throw new AuthenticationException( message: 'Invalid token');
    }
    $header = json_decode(base64_decode($parts[0]), associative: true);
    $headers = (object)$header['alg'];
    try {
        $decodedToken = JWT::decode($a, $this->getKey(), &: $headers);
        $roleArray = $decodedToken->resource_access->{'l[redacted]t'}->roles;
        $this->isAdmin = in_array( needle: 'l[redacted]n', $roleArray);
    } catch (\Exception $e) {
        throw new AuthenticationException($e->getMessage());
    }

    return new SelfValidatingPassport(
        new UserBadge($decodedToken->sub, function (string $accountId) {
            $account = $this->accountRepository->find($accountId);
            if (null === $account) {
                $account = new Account($accountId);
                if ($this->isAdmin) {
                    $account->setRoles(['ROLE_USER', 'ROLE_ADMIN']);
                }

                $this->entityManager->persist($account);
                $this->entityManager->flush();
            }

            return $account;
        })
    );
}
```

Obrázek 23 Služba pro autentikaci (vlastní zdroj)

Návratová hodnota této funkce je podle interface Passport, takže vrátím svůj validační passport s vlastní logikou. Vytvořil jsem nový UserBadge, kde jsem jako parametr vložil Uuid z tokenu a jako druhý funkci. V této funkci jsem vytvořil podmínku, pokud uživatel se

stejným id existuje, tak vratím účet uživatele. Pokud neexistuje, což znamená, že se do aplikace přihlašuje poprvé, bude jeho účet vytvořen a přiřazeny práva podle interního systému. Jak jsem již psal, budu ukládat pouze nutné informace, v tomto případě pouze ID, které je ve formátu Uuid, tedy id shodné s tím, jaké má u uživatele uložené služba keycloak a vůči kterému se pak uživatel ověřuje v jeho existenci v databázi.

Jako poslední jsem dopsal kód k metodám, kdy autentizace projde. Zde jsem jen nechal pokračovat autentizaci. Pokud se autentizace nepodaří, vyhodím chybu ve formátu JsonResponse s chybou a statusem 401 – Unauthorized.

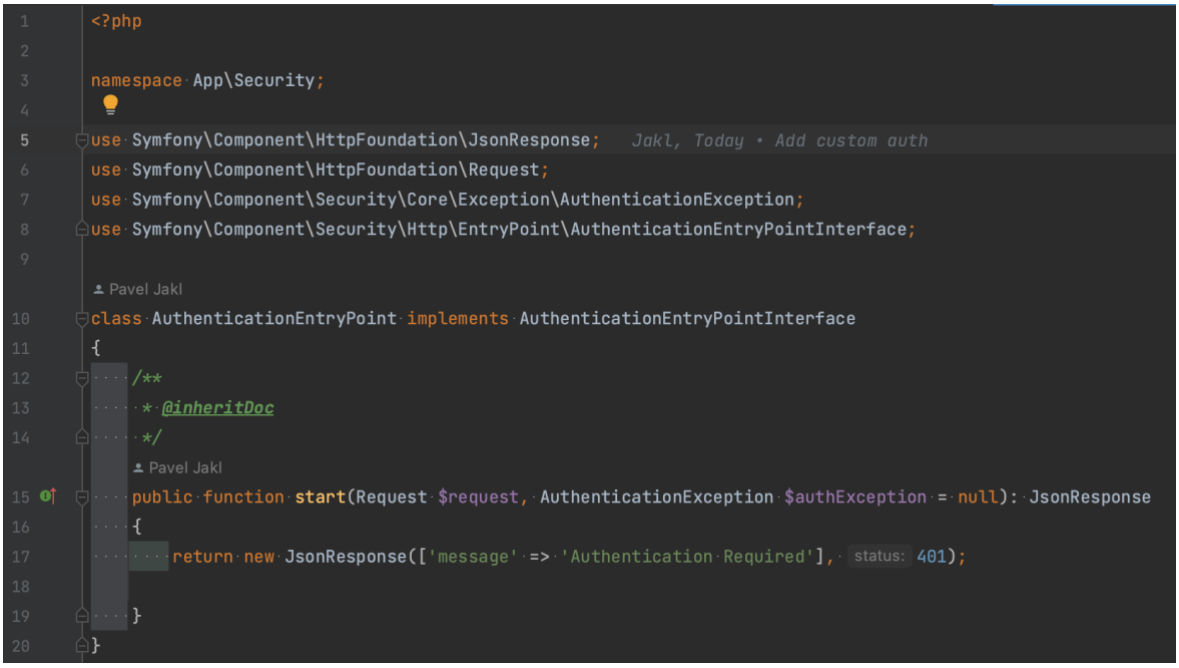
Aby toto zabezpečení fungovale na api, přidal jsem ještě do konfiguračního souboru několik změn. Nejprve jsem povolil authenticator_manager, abych mohl využít svou definovanou službu. Poté definovat providera, což v mém případě je entita account a jako hlavní properta id. Do firewall sekce jsem přidal parametr custom_authenticators, kterému jsem předal službu. Jakom poslední jsem specifikoval přístup, které adresy budou veřejné, které jsou pouze po úspěšné autentizaci a které mají ještě k tomu zabezpečení v podobě admin role.

Již dříve jsem specifikoval dokumentaci api v nastavení projektu a na adrese /api/docs běží swagger pro testování api. Tuto adresu jsem zatím nechal veřejnou, ale v budoucnu bude i ta pod plnou autentizací, jelikož celé api je zabezpečeno, a tudíž by neměl nikdo dostat přístup ani pro procházení dokumentace.

```yaml
security:
    enable_authenticator_manager: true
    providers:
        accounts:
            entity:
                class: App\Entity\Account
                property: id
    firewalls:
        dev:
            pattern: ^/_(profiler|wdt)
            security: false
        main:
            lazy: true
            entry_point: App\Security\AuthenticationEntryPoint
            custom_authenticators:
                - App\Security\KeycloakAuthenticator

    access_control:
        - { path: ^/api/docs, roles: PUBLIC_ACCESS }
        - { path: ^/api, roles: IS_AUTHENTICATED_FULLY }
        - { path: ^/api/products, roles: [IS_AUTHENTICATED_FULLY, ROLE_ADMIN] }
```

Obrázek 24 Zabezpečení API (vlastní zdroj)

Jako poslední část, jsem musel vyřešit chybovou hlášku, když se někdo pokusí přistoupit k zabezpečenému api. Pro tento účel jsem zvolil AuthenticationEntryPoint na základě jednoduché implementace a funkčnosti. Je to nativní součást symfony, takže nebyl žádný problém s vytvořením.



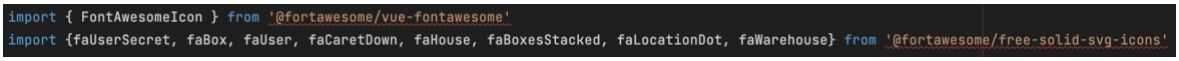
Obrázek 25 Definice vstupního bodu pro API (vlastní zdroj)

Tato služba importuje interface se stejným názvem a má jednu metodu start. Do této metody jsem vložil jednoduše jen odpověď na to, když někdo přijde na api bez svého tokenu.
Aby aplikace ověřila při každém dotazu, zda je uživatel autentizovaný, přidal jsem parametr entry_point do firewall/main.

4.5 Frontend část

4.5.1 Základní konfigurace aplikace

Pro vytvoření aplikace jsem použil oficiální příkaz *vue create matemanagerfe*. Tento příkaz mě přesměruje do vue CLI console, kde si mohu vybrat, co všechno potřebuji do nového projektu nainstalovat. Zvolil jsem vue3, prettier, vue-router, babel, pinia (local storage) a základní integraci testů. Projekt je psaný v composition API s využitím typescriptu.
Celý projekt využívá ikony z FontAwesome, které je třeba definovat v main.ts. FontAwesome jsem nainstaloval pomocí npm a importoval. Všechny ikony je třeba explicitně importovat z knihovny.



Obrázek 26 Import ikon pro aplikaci (vlastní zdroj)

Následně je potřeba dát aplikaci vědět o tom, že má využívat definovaný import. Této skutečnosti se docílí pomocí deklarace .component(…). Skrze celou aplikaci tedy budou dostupné předem definované ikony, které aplikace při svém buildu zahrne.

Výsledný soubor s inicializací a základním nastavením projektu vypadá následovně:

```
import '@/assets/sass/main.scss'
import 'bootstrap'

import { createApp } from 'vue'
import { createPinia } from 'pinia'
import { library } from '@fortawesome/fontawesome-svg-core'
import { FontAwesomeIcon } from '@fortawesome/vue-fontawesome'
import {faUserSecret, faBox, faUser, faCaretDown, faHouse, faBoxesStacked, faLocationDot, faWarehouse} from '@fortawesome/free-solid-svg-icons'

import App from './App.vue'
import router from './router'
import KeyCloakService from "@/security/KeycloakService";



library.add(faUserSecret, faBox, faHouse, faUser, faCaretDown, faBoxesStacked, faLocationDot, faWarehouse);
export const initializeApp = () => {
    createApp(App)
        .use(createPinia())
        .use(router)
        .component('font-awesome-icon', FontAwesomeIcon)
        .mount("#app");
}
  Jakl, 07.10.2023, 13:04 • Initial commit
KeyCloakService.CallLogin(initializeApp)
```

Obrázek 27 Inicializace celé aplikace Vue.js (vlastní zdroj)

Aplikace je inicializovaná do konstanty initializeApp, kde se volá arrow function s definicí, co všechno má aplikace při jejím vytvoření inicializovat. Zde je to Pinia, vue-router pro vytvoření route, font-awesome knihovna a jako poslední se aplikace připne na zvolené ID containeru.

4.5.2 Lokální úložiště

Pro potřeby uložení informací o aktuálním přihlášeném uživateli jsem definoval lokální úložiště pomocí knihovny Pinia. Tuto možnost jsem zvolil na základě pozitivních zkušeností s touto knihovnou a na rozdíl od jiných knihoven, například VUEX není tak robustní, a i tak nabízí všechny potřebné funkce pro práci s lokálním úložištěm prohlížeče. Vytvořil jsem tedy soubor keycloak.ts, kde jsem definoval konstanty, které bude úložiště zpracovávat. Pro deklaraci úložiště je třeba využít **defineStore**, kde jsem funkci předal dva parametry a to jméno úložiště a funkci, ve které definuje dvě konstanty:

- keycloak
- isAdmin

Tyto dvě konstanty stačí k tomu, abych byl schopen získat informaci o uživateli. Respektive by stačila je konstanta keycloak, isAdmin je konstanta, ve které si uchovávám bool hodnotu, zda je uživatel v roli administrátora, nebo jen v roli uživatele. Na základě toho mohu jednoduše ovlivnit, zda se určité komponenty, které jsou zpřístupněny pouze administrátorům, se budou zobrazovat v DOMu či nikoli.

```
import { ref } from 'vue'
import { defineStore } from 'pinia'
import type Keycloak from "keycloak-js";

export const useKeycloak = defineStore('keycloakStore', () => {
    const keycloak = ref(null as Keycloak | null)
    const isAdmin = ref<boolean>()

    return { keycloak, isAdmin }
})
```

Obrázek 28 Definice úložiště pro data z autentizace (vlastní zdroj)

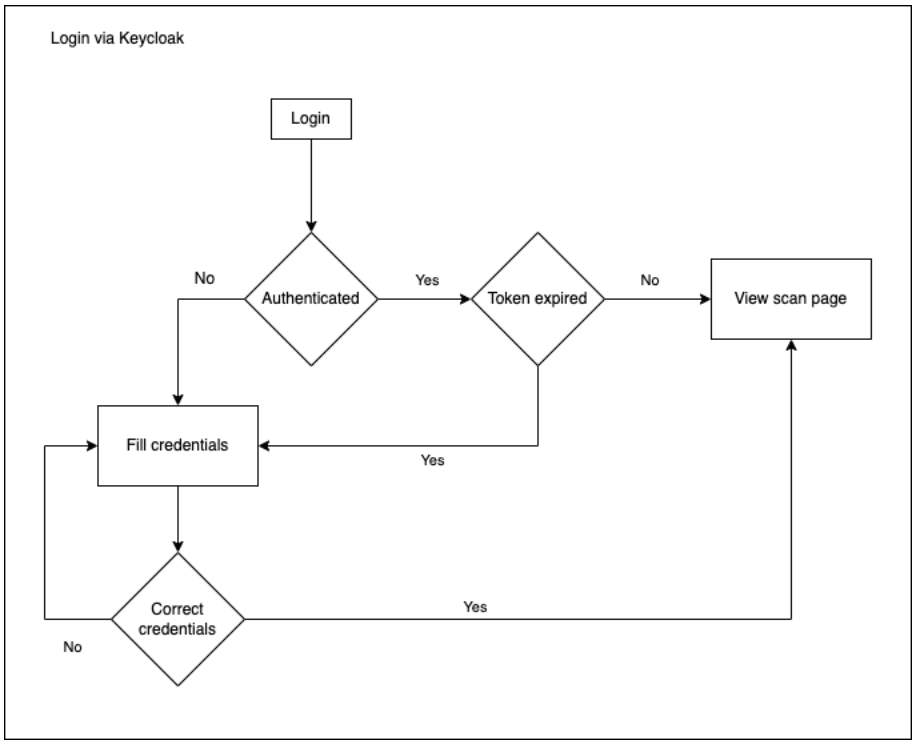
Zde je výsledná konfigurace lokálního úložiště. Využívám export rovnou při deklaraci konstanty, takže již nemusím dávat informaci o tom, že se musí exportovat. Jednotlivé konstanty vracím ještě ve funkci, aby se dali využít samostatně a mohl jsem si tak vybrat specifickou hodnotu.

4.5.3 Přihlášení a autorizace

4.5.3.1 User flow diagram

Následující user flow diagram znázorňuje průchod přihlašováním. Poté, co uživatel otevře stránku se kontroluje, zda je uživatel již přihlášený, tedy, jestli má již vygenerovaný token. Pokud token není expirovaný, bude přesměrovaný na stránku s naskenováním QR kódu. Pokud je token expirovaný, bude uživatel přesměrovaný na stránku přihlášení. Stejně tak, pokud uživatel nebude mít token. Na přihlašovací stránce vyplní jméno a heslo. Pokud jsou

údaje správné, bude přesměrován na stránku s naskenováním. Pokud ne, bude se akce opakovat do doby, kdy nebudou správně zadány informace pro přihlášení.



Obrázek 29 Flow diagram přihlášení pomocí Keycloak (vlastní zdroj)

4.5.3.2 Keycloak služba a inicializace

Jelikož je celá aplikace zabezpečená autentizací oproti internímu firemnímu systému, který běží pod VPN, zvolil jsem na základě již existující konfigurace prostředí službu třetí strany Keycloak, která poskytuje veškeré nastavení pro využití v této aplikaci.

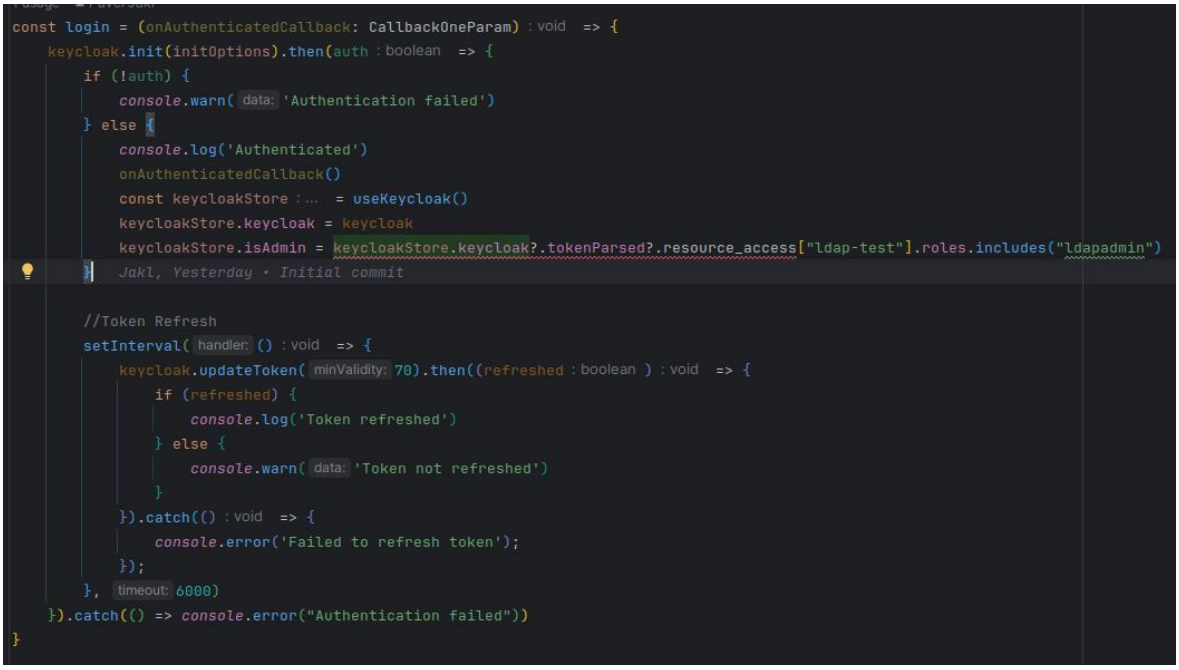
Jako první jsem pomocí NPM manageru nainstaloval knihovnu keycloak-js. Tato knihovna obsahuje potřebný JS adaptér, nad kterým jsem vytvořil základní konfiguraci pro přihlášení a ověřování vůči firemnímu systému.

Jako první bylo třeba nastavit primární přístupové parametry konfigurace pro instanci Keycloaku a to url adresu, na které běží keycloak, realm a clientId. Tyto parametry jsou vytvořeny vždy při zakládání nového realmu v administraci. Dále jsem definoval dodatečné chování – v tomto případě při načtení vyžadovat login. Jelikož využívám TS, je zde vytvořený interface pro callback funkci, pomocí které proběhne autentizace.

Obrázek 30 Inicializace služby keycloak v aplikaci (vlastní zdroj)

Konstanta login se stará o samotný proces autentizace. Jedná se o arrow function, která bere na vstupu parametr již zmiňovaný interface. Následně proběhne inicializace instance keycloaku s parametry a poté se zavolá funkce auth. Na obrázku jsou ponechány výpisy a upozornění do developer konzole pro lepší debugování a porozumění průchodu kódem. Pokud uživatel není autentizovaný, konzole vypíše error v podobě chyby autentizace. Pokud ano, konzole vypíše, že je uživatel autentizovaný. Poté se vytvoří instance Keycloak store (Pinia), což je lokální úložiště. Do této instance se uloží objekt, který vrátí keycloak a dále do lokálního úložiště uložím informaci o tom, zda je daný uživatel oprávnění k tomu, aby mohl dělat úpravy v administraci. O to se stará konstanta isAdmin, která má hodnotu boolean. Následně proběhne aktualizace tokenu, pokud je to potřeba. Zde jsou také ponechány logy pro lepší debuging aplikace. Pokud byl token aktualizovaná, vypíše se do konzole, stejně tak, pokud došlo k chybě nebo token aktualizovaný nebyl. Zde je nastavený interval na 6000ms.

Pokud se vytvoření instance z nějakého důvodu nepodaří, zachytí se chyba a následně bude vypsána do konzole. V produkčním prostředí bude tato informace předána uživateli v podobě error zprávy.

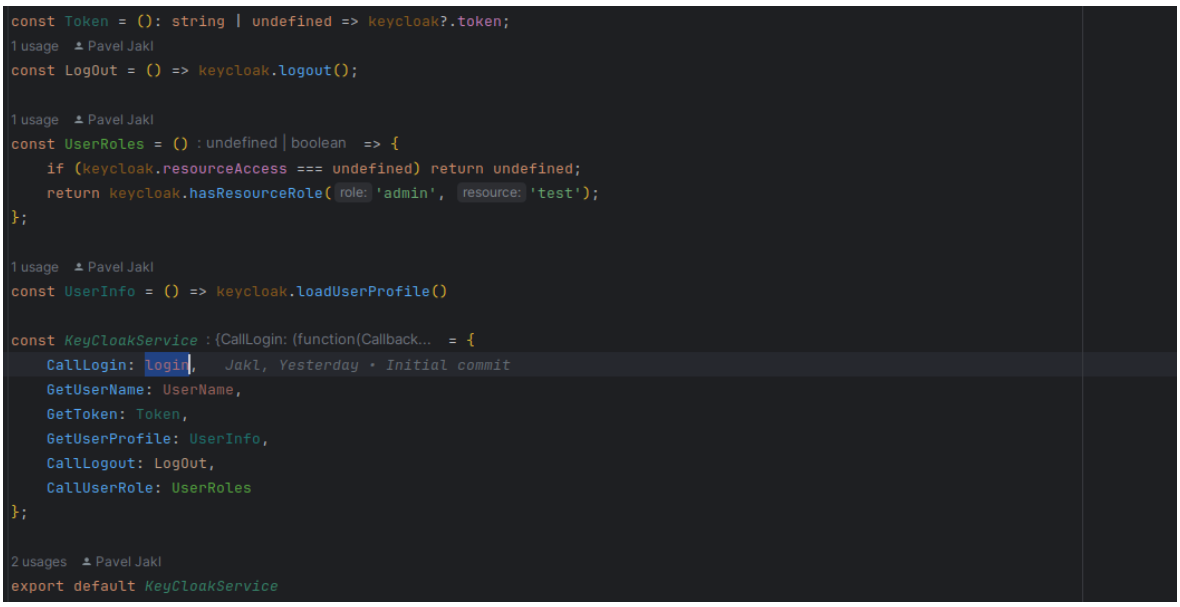


Obrázek 31 Konstanta login a její vnitřní konfigurace (vlastní zdroj)

Následně jsem definoval několik konstant, jako arrow functions pro využití skrz celou aplikaci, jako je například konstanta UserName, Token, LogOut, UserInfo. Tyto konstanty si sahají do lokálního úložiště k objektu keycloak a vrací specifikované informace.

Jako poslední byla definována jedna konstanta jako objekt, která obsahuje všechny informace, které budu využívat v komponentách aplikace.

Jelikož tento soubor je vytvořený jako služba, je nutné exportovat výslednou konstantu, abych ji mohl použít mimo rozsah tohoto souboru.

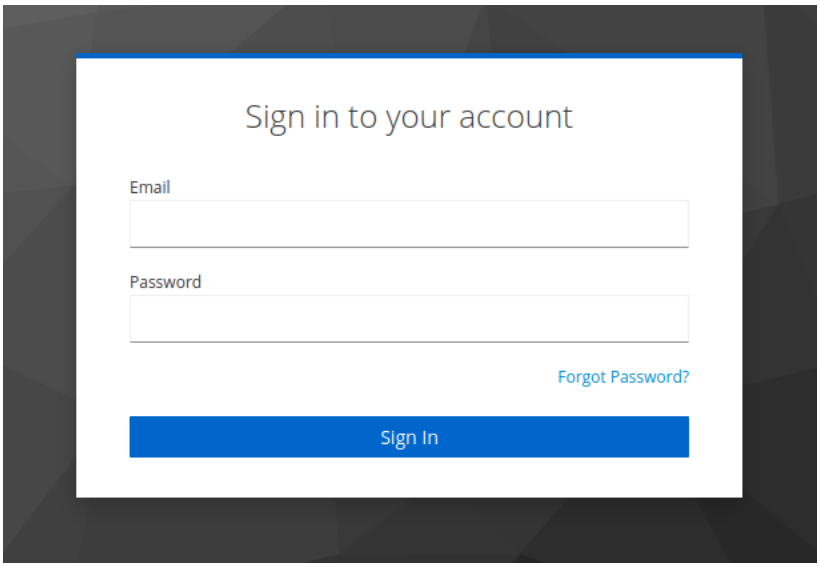


Obrázek 32 Vrácení funkcí vytvořenou službou (vlastní zdroj)

Tuto službu použiji při vytváření instance Vue.js. Nejprve je vytvořena konstanta, kde se inicializuje instance. Vytvoří se aplikace s danou konfigurací. Tato konstanta je také jako funkce a následně se vytvořená aplikace předá do funkce CallLogin, kterou jsem definoval ve službě KeCloakService. Takže po spuštění serveru se inicializuje a vytvoří aplikace. Po vytvoření je uživatel rovnou přesměrován na login stránku keycloaku, kde zadá jméno a heslo. Po úspěšném přihlášení je autentizovaný a má přístup do aplikace.

```
export const initializeApp = () : void  => {
    createApp(App)
        .use(createPinia())
        .use(router)
        .component( name: 'font-awesome-icon', FontAwesomeIcon)
        .mount( rootContainer: "#app");
}

// initializeApp()   Jakl, Yesterday • Initial commit


KeyCloakService.CallLogin(initializeApp)
```

Obrázek 33 Inicializace služby při mount aplikace (vlastní zdroj)



Obrázek 34 Přihlašovací obrazovka služby KeyCloak (vlastní zdroj)

4.5.3.3 JWT token

Keycloak vytvoří JWT token, který obsahuje všechny potřebné informace pro to, aby mohl uživatel komunikovat bezpečně s backendem a tak bez problému užívat celou aplikaci. Tento token je důležitou součástí aplikace, jelikož na jeho základu je uživatel autorizován a

autentizován. Určuje tedy, zda je uživatel vůbec schopen se dostat do aplikace (to záleží na nastavení realmu keycloaku) a jaké má práva v rámci aplikace. Tyto role jsou nastaveny ve firemní interní aplikaci pro každého zaměstnance. Plus tento token obsahuje dodatečné firemní informace, na základě, kterých má uživatel právo na aktivity v rámci aplikaci. Tento token se bude posílat s každým požadavkem na server a bude sloužit jako přístupový token k privátnímu API, které poskytuje backend. Tento token jsem v aplikaci exportoval v rámci KeyCloakService, viz. výše, aby mohl být zavolán jednoduše z jakékoli části aplikace a aby byl i po jeho změně dostupný a funkční pro komunikaci. Token používá šifrování SHA256.

4.5.3.4 Vue router – vytvoření routes

Aplikace musí mít možnost vytvářet adresy, na kterých bude zobrazen obsah. Pro tento účel jsem zvolil oficiální balíček vue-router, který implementuje toto řešení. Pomocí vue-router jsem nadefinoval adresy, které jsou potřeba pro běh aplikace. Routy se vytváři jednoduše pomocí následujícího schématu.

```
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'scan',
      meta: {
        isAuthenticated: true
      },
      component: () => import('../views/ScanView.vue'),
    },
    {name: 'locality'...},
    {name: 'admin'...}
  ]
})

router.beforeEach((to, from, next) => {...})
export default router
```

Obrázek 35 Rozdělení route v aplikaci (vlastní zdroj)

Jako první deklaruji konstantu router, která má v sobě funkci createRouter, která má jako parametr objekt, který má parametr history, jenž definuje základní url aplikace a parametr routes, kde defunuji konkrétní routy. Routes je pole objektů route, kdy každý objekt má parametr path – url adresa, name – pojmenování v rámci terminologie aplikace a využití v komponentách, dále meta parametr, který říka, že daná routa má určitě označení, v tomto případě, že je přístupná pouze po přihlášení (isAuthenticated) a nakonec component, kde definujeme soubor (view), ze kterého si bude brát data. Tento zápis komponenty určuje, že daná komponenta je takzvaně lazy load, což znamená, že se načte, až když je daní adresa uživatel vyžádána a nebude se načítat při prvním stažení stránky, což ušetří velikost stahového obsahu, a tudíž zlepší prvotní načtení aplikace, než se obsah dostane do http cache. Následnějsme exportoval constantu router, abych ji mohl využívat v rámci celé aplikace.
V rámci aplikace jsem vyřešil také jednotlivá oprávnění uživatelů. Tyto práva jsou z již zmíněné autentizace pomocí keycloaku a získám je pomocí lokálního úložiště prohlížeče. Další krok tedy byl, vytvořit routes, které jsou dostupné pouze uživatelům s rolí administrátora. Jelikož těchto route bude více a mají stejnou rodičovskou stránku, vytvořil jsem jenu rodičovskou routu, které jsem přiřadil její potomky. Což znamená, že všechny potomci dědí stejné nastavení, jako má rodičovská routa. Rodičovskou routu jsem zabezpečil pomocí meta parametu isAdmin. Tento parametr je nastaven na true, což znamená, že uživatel, který nemá roli administrátora se na danou stránku nedostane. Deklarace je stejná,

jako definici běžné routy, jen je přidán parametr children, což je znovu pole objektů route. Výsledná konfigurace vypadá následovně:

```
{
  path: '/admin',
  name: 'admin',
  meta: {
    isAuthenticated: true,
    isAdmin: true,
  },
  component: () => import('../views/admin/AdminView.vue'),
  children: [
    {
      path: 'dashboard',
      name: 'dashboard',
      component: () => import('../views/admin/DashboardView.vue')
    },
    {
      path: 'products',
      name: 'products',

      component: () => import('../views/admin/ProductView.vue')
    },
    {
      path: 'products',
      name: 'products',
      component: () => import('../views/admin/ProductView.vue')
    },
```

Obrázek 36 Admin route rozdělení (vlastní zdroj)

4.5.4 Objednávka

Po přihlášení do administrace jsem automaticky přesměrován na stránku, kde se mi zobrazí možnost naskenovat QR kód, který je na některé z lokalit. Tento způsob má svá jistá omezení a naráží tak na možnost rozšíření aplikace, a to vytvoření mobilní aplikace, která daleko lépe pracuje s oprávněním přístupu k fotoaparátu.

QR kód pro danou lokalitu je generovaný v administraci, automaticky na základě názvu kategorie a jejího ID. Do QR kódu je pak vnesena URL adresa s ID lokality, kam jsem přesměrován po naskenování QR kódu.

Po naskenování kódu jsem přesměrován na přehled všech produktů, které jsou na dané lokalitě k dispozici. Jako uživatel mám možnost vybrat si produkty, které jsou k dispozici pomocí výběru počtu u každého z nich. Po výběru produktů se všechny vybrané produkty a jejich množství uloží do košíku, který reprezentuje useCartStore.

```
import { defineStore } from 'pinia';

export const useCartStore = defineStore( id: 'cart', options: {
    state: () => ({
        cartItems: []
    }),
    actions: {
        addProduct(product) {
            this.cartItems.push(product);
        }
    }
});
```

Obrázek 37 Vytvoření úložiště pro objednávku (vlastní zdroj)

useCartStore je instance pinii, což je tedy lokální úložiště. V tomto případě kód uchovává po dobu opětovného načtení stránky stav vybraných produktů a pokud jsou nějaké vybrané dodatečně, přidá je k již existujícím produktům.

Jako následný krok je zobrazení všech vybraných produktů, které jsem si zvolil. Jedná se o shrnutí objednávky, kde je přehled všech produktů, jejich počet a cena za každý produkt. Je zde uvedena také celková cena za všechny produkty. V této části je vše připraveno pro vytvoření objednávky.

Po kliknutí na vyzvednout se vytvoří objednávka, kde se uvede id uživatele, který si produkty objednal, dále produkty, které jsou uloženy v košíku, se uloží jako orderItems do relační tabulky Order, datum vytvoření objednávky, datum splatnosti a celková částka za produkty. Tímto je vyzvednutí produktů uživatelem dokončeno. Já, jako uživatel systému si pak mohu ve svém profilu zobrazit všechny mé objednávky a mohu také vidět stav zaplacení. Pokud jsou zaplacené, je u nich zelená fajfka. Pokud ne, je u objednávky zobrazená červený křížek.

4.5.5 Profil uživatele

Na profilu uživatele je k dispozici několik přehledů. Jelikož je celý účet navázaný na lokální interní systém správy uživatelů, neumožňuje provádět změnu jména nebo profilové fotky.
Na profilu je možnost procházet již vytvořené objednávky s jejich statusem o zaplacení, dále také celkovou částku, která je nutná uhradit. Pokaždé, když je uživatelem vytvořená objednávka, propíše se do BE části.

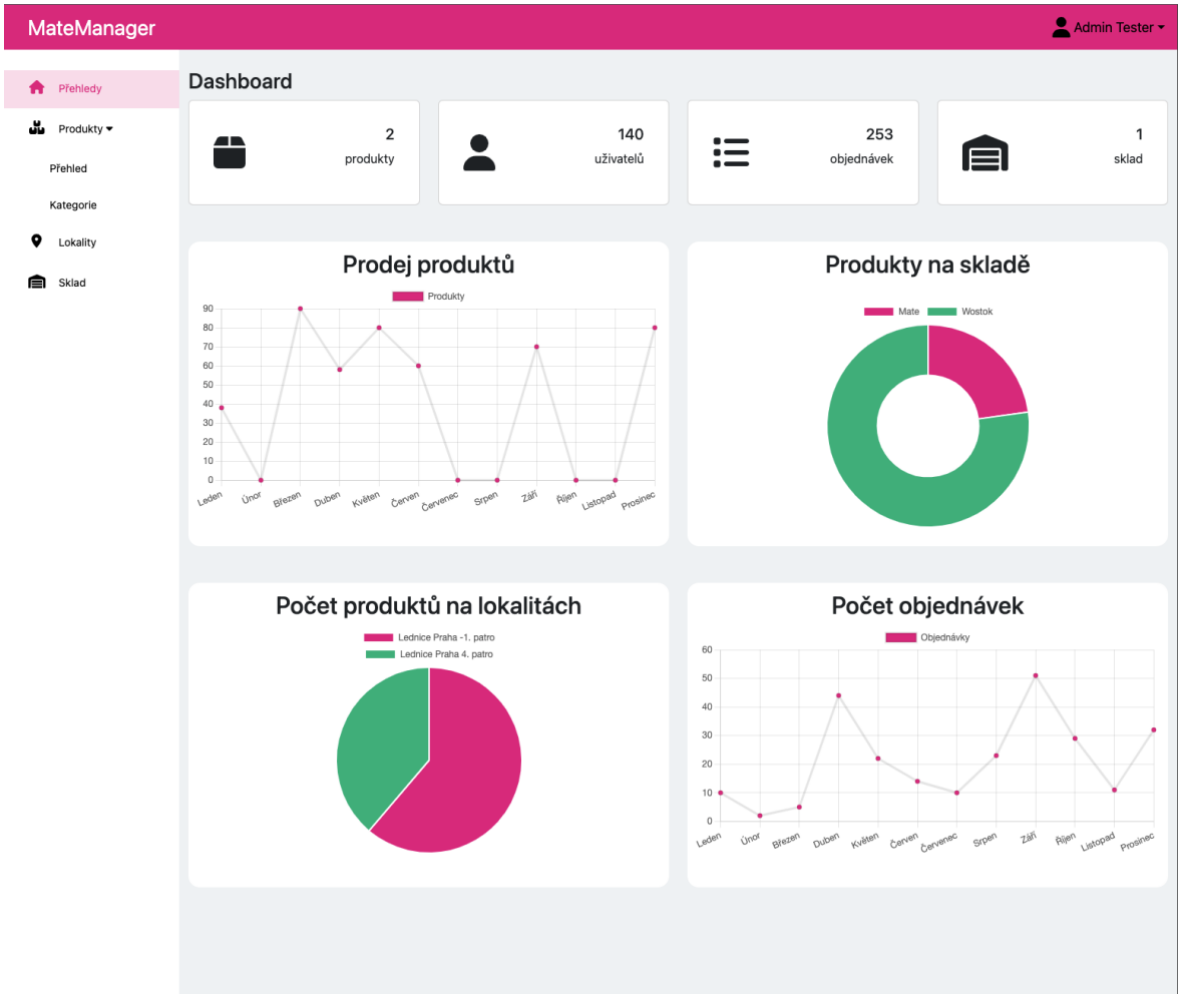
4.5.6 Upozornění o platbě

Vždy ke konci měsíce přijde uživateli upozornění do firemního emailu o budoucí platbě za daný měsíc. K tomuto na BE částí slouží cron jobs, který při definovaném datu spustí skript, který vezme daného uživatele, projde všechny jeho objednávky a vytvoří nový záznam do tabulky monthlyPayments, kde je zapsané id uživatele, celková částka a datum splatnosti. Následně tento záznam zobrazí FE část v sekci platby. U každé platby bude uvedená QR kód, na který účet se má platba zaslat.

4.5.7 Administrace

4.5.7.1 Nástěnka

Na hlavní straně administrace je vytvořen celkový přehled, kde může administrátor rychle vidět stav produktů. Na nástěnce je zobrazen celkový počet produktů, počet objednávek, nezaplacené objednávky, počet všech uživatelů systému a stav jednotlivých skladů, kde je hned vidět, zda produkty na skladu jsou, dochází anebo vůbec nejsou skladem.
Z nástěnky se administrátor dostane přímo na jednotlivé ukazatele a je to tak jednodušší než se dostávat na specifickou stránku přes hlavní navigaci. Tato nástěnka se po testovacím období může změnit nebo se přijde na to, že jsou třeba jiné ukazatele. Zde je prostor také pro vytvoření nástěnky, kterou si bude administrátor personalizovat. Všechna data, která se nachází v přehledu nebo v grafech na nástěnce jsou v reálném čase, tudíž pokud vytvořím objednávku, okamžitě se aktualizuje stav na nástěnce, změní se počet produktů na lokalitě a ve skladu, přibude objednávka a grafy, které zobrazují měsíční přehledy budou tuto skutečnost reflektovat po měsíční aktualizaci dat.



Obrázek 38 Přehled pro administrátory (vlastní zdroj)

4.5.7.2 Modul skladu

Uživatel, který má práva na spravování aplikace, obsluhuje sklad. Modul skladu vychází z jednoduchého modelu, kdy jsou vytvořeny produkty, které chci v rámci celého skladu evidovat. V sekci sklad si následně vytvořím instance, kde chci evidovat jejich množství. V tomto případě jsem vytvořil sklad Praha, kde budu chtít evidovat produkty pro dané lokality. Po kliknutí na detail skladu mohu přes tlačítko přidat produkty. Zvolím si již existující produkt a jeho množství, které chci naskladnit. V rámci první verze nejsou řešeny dodavatelé a jejich správa. Produkty se nakoupí a jejich výsledný počet je následně zapsán do skladu. Pokud jsou některé produkty poškozené, je zde možnost odepsání produktů přes sekci odpisů, kde se zvolí produkt a množství které je třeba odepsat. Následně se upraví celkový počet produktů ve skladu v závislosti na množství odepsaného produktu. Tyto operace jsou viditelné v sekci sklad/odpisy. Takto jsou evidované všechny produkty na daném skladu. V rámci konkrétního skladu je také možnost kontrolovat pohyby, které byly

v rámci skladu vykonány. V této sekci je teda možnost vidět, které produkty byli naskladněny, které produkty byly odepsány a které přesunuty na danou lokalitu. V rámci záznamu jsou evidovány položky:

- Produkt, kterého se akce týká
- Množství, které bylo v rámci akce vybráno
- Zda se jedná o naskladnění, vyskladnění nebo odpis
- Datum, kdy byla provedena akce

4.5.7.3 Objednávky

Objednávky v rámci administrace jsou viditelné jen pro uživatele, kteří mají přístup, tedy roli administrátora. Všechny objednávky, které jsou vytvořeny, jsou zobrazené v administraci v sekci Objednávky. U všech objednávek je evidovaný stav, zda jsou zaplaceny nebo ne. U všech objednávek jsem vytvořil indikaci stavu o zaplacení, která bude rychle vyjadřovat stav objednávky a zda daný uživatel už zaplatil, či nikoli. Po zobrazení detailu objednávky je vidět, kdo objednávku udělal, produkty, které jsou v objednávce uvedeny, celková částka, která je nutná zaplatit, informace, zda objednávka byla zaplacena a datum, kdy byla objednávka vytvořena. Také je zde informace, zda byla odeslána výzva k platbě.

Pokud nebude objednávka zaplacena následující měsíc, přijde znovu upomínka k platbě. V první verzi jsem zvolil postup takový, že objednávka bude obsahovat pole s informací, zda byla částka zaplacena či nikoli a pokud nebude uhrazena následující měsíc, je zde možnost znovu zaslat upomínku na email.

Jelikož v první verzi není napojení na platební bránu a není zajištěno propojení s bankovnictvím, musí se platby evidovat ručně, tudíž administrátor, který řeší platby (někdo, kdo pracuje na BO a má přístup k podnikovému účtu) ověří, zda platba proběhla v pořádku a připsala se na účet. Tato částka se dá spárovat s příchozí platbou pomocí variabilního kódu, který bude vždy u každé objednávky vytvořen.

4.5.8 Analýza existujících dat

K dispozici jsem dostal data o prodeji limonád za posledních pár let. Data nejsou úplná, respektive jsou evidovaná náhodně, a ne jednou měsíčně. K analýze jsem si vybral nejvíce prodávaný produkt za poslední dva roky. Jedná se o limonádu Wostok.

4.5.8.1 Wostock

Jako vzorek byl použitý záznam z posledních tří let prodeje daného produktu. Na těchto datech jsem analyzoval základní statistické ukazatele a zda jsou data normálně rozdělena.

Základní statistické ukazatele

Tabulka 2 znázorňuje data, která jsem dostal k dispozici. Na první pohled je vidět, že data nejsou celistvá a obsahují chybějící hodnoty, jelikož se zapsání prodaných produktů neprovádělo každý měsíc. Základní analýzu dat provedu na tomto setu dat a později doplním data, která chybějí.

Datum	Wostock
2021-01-27	286
2021-05-10	462
2021-09-16	349
2021-11-19	86
2022-01-01	38
2022-03-10	90
2022-03-22	13
2022-04-21	58
2022-05-13	80
2022-06-17	66
2022-09-29	70
2023-01-03	251
2023-02-03	118
2023-06-13	60

Tabulka 2 Základní data (vlastní zdroj)

Tabulka 3 představuje pokročilou statistickou analýzu proměnné 'Wostock'. Sada dat se skládá z 14 pozorování s průměrnou hodnotou 25.65 a standardní odchylkou 23.66, což naznačuje vysokou variabilitu v rámci sady dat. Šikmost 1.12 ukazuje na asymetrické rozdělení z pravé strany, zatímco negativní hodnota špičatosti -0.36 naznačuje mírně plošší rozdělení, než je normální rozdělení. Koeficient variace je vysoký, činí 92.28 %, což signalizuje, že standardní odchylka je velká ve srovnání s průměrem. Standardní chyba průměru je 6.32, což ukazuje na přesnost odhadu průměru. Součet čtverců bez korekce a s korekcí poskytuje informace o celkové variabilitě v datech a je základem pro výpočet variance, která je 559.94, což dále potvrzuje vysokou rozptylnost v datech.

Dále jsem testoval, zda jsou data normálně rozložena. Pro toto testování bylo využito několik testů, ale primárně jsem přihlédl ke dvou a to Shapiro-Wilkův test a Kolmogorov-Smirnovův test.

Variable: Wostock (Wostock)

Moments			
N	14	Sum Weights	14
Mean	25.6428571	Sum Observations	359
Std Deviation	23.6630421	Variance	559.93956
Skewness	1.12139782	Kurtosis	-0.3564988
Uncorrected SS	16485	Corrected SS	7279.21429
Coeff Variation	92.2792727	Std Error Mean	6.32421401

Tabulka 3 Statistická data pro analýzu (vlastní zdroj)

Tabulka 4 prezentuje výsledky statistických testů normálnosti provedených na analyzovaném datovém souboru. Shapiro-Wilkův test poskytl statistiku W ve výši 0.781245 s p-hodnotou 0.0030, což naznačuje, že data nejsou normálně rozdělena na hladině významnosti 0.05. Kolmogorov-Smirnovův test, s hodnotou D 0.269505 a p-hodnotou menší než 0.0100, dále potvrzuje odchylku od normálnosti. Tyto výsledky konzistentně naznačují, že předpoklad normálního rozdělení dat není splněn.

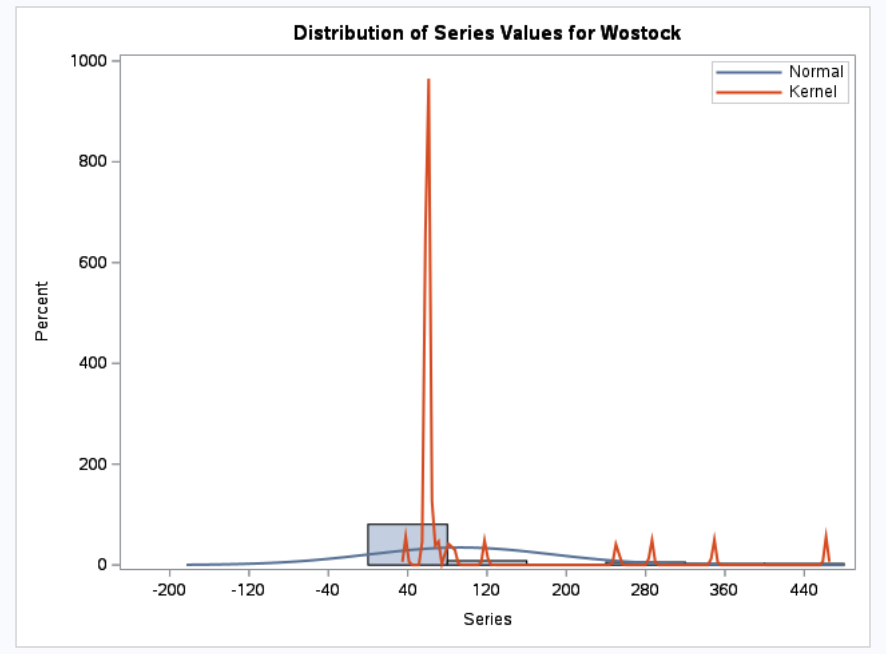
Tests for Normality				
Test	Statistic		p Value	
Shapiro-Wilk	W	0.781245	Pr < W	0.0030
Kolmogorov-Smirnov	D	0.269505	Pr > D	<0.0100
Cramer-von Mises	W-Sq	0.230722	Pr > W-Sq	<0.0050
Anderson-Darling	A-Sq	1.312235	Pr > A-Sq	<0.0050

Tabulka 4 Testy normality (vlastní zdroj)

Jak je vidět v tabulce 2, jsou zde chybějící data, respektive data nejsou rozdělena po měsíci, jelikož nebyl vytvořen každý měsíc záznam. Upravil jsem tedy tabulku, která obsahuje všechny měsíce od roku 2021. Je jasné, že se změní její statistické ukazatele, ale po opětovném výpočtu stále platí, že data nejsou normálně rozložena. Chybějící hodnoty jsem nahradil poslední nechybějící hodnou akumulované časové řady pro každý chybějící řádek. Jejich hodnota je 60, ale dá se předpokládat, že hodnoty, které chybí, jsou zahrnuty v následujícím měsíci po chybějících hodnotách.

	Mesic	Wostock
1	JAN2021	286
2	FEB2021	60
3	MAR2021	60
4	APR2021	60
5	MAY2021	462
6	JUN2021	60
7	JUL2021	60
8	AUG2021	60
9	SEP2021	349
10	OCT2021	60
11	NOV2021	86
12	DEC2021	60
13	JAN2022	38
14	FEB2022	60
15	MAR2022	60
16	APR2022	58
17	MAY2022	80
18	JUN2022	66
19	JUL2022	60
20	AUG2022	60
21	SEP2022	70
22	OCT2022	60
23	NOV2022	60
24	DEC2022	60
25	JAN2023	251
26	FEB2023	118
27	MAR2023	60
28	APR2023	60
29	MAY2023	60
30	JUN2023	60
31	JUL2023	60
32	AUG2023	60
33	SEP2023	60
34	OCT2023	60
35	NOV2023	60
36	DEC2023	60

Tabulka 5 Upravená data (vlastní zdroj)



Graf 1 Distribuce hodnot pro sérii Wostok (vlastní zdroj)

Nad takto upraveným setem dat provedu analýzu časové řady a určím predikci objemu do budoucna. Data byla upravena pro využití v časové řadě pomocí programu Sas, konkrétně pomocí Data Preparation.

Graf 1 ilustruje distribuci hodnot série pro proměnnou Wostock, přičemž histogram odhaluje značnou koncentraci hodnot blízko nuly a několik významných výkyvů, které představují možné anomálie. Překryv normální rozdělovací křivky není vhodným reprezentantem distribuce, což je zřejmé z nedostatečné shody s daty. Jádrový odhad hustoty

pravděpodobnosti nabízí alternativní, ne-parametrický pohled na data, avšak ani tento odhad nezachycuje distribuci přesně kvůli výrazným špičkám, které jsou v histogramu přítomny. Tato nesrovnalost mezi odhady hustoty a skutečnou distribucí naznačuje, že by bylo vhodné data dále analyzovat, zejména s ohledem na identifikaci a příčiny těchto anomálií.
Na základě toho výsledku jsem změnil chybějící hodnoty a dal jsem jim hodnotu 0. Provedl jsem znovu analýzu základních ukazatelů.

Moments			
N	36	Sum Weights	36
Mean	22812.5	Sum Observations	821250
Std Deviation	320.463637	Variance	102696.943
Skewness	0.00042493	Kurtosis	-1.2000541
Uncorrected SS	1.87384E10	Corrected SS	3594393
Coeff Variation	1.40477211	Std Error Mean	53.4106062

Tabulka 6 Ukazatele pro upravená data (vlastní zdroj)

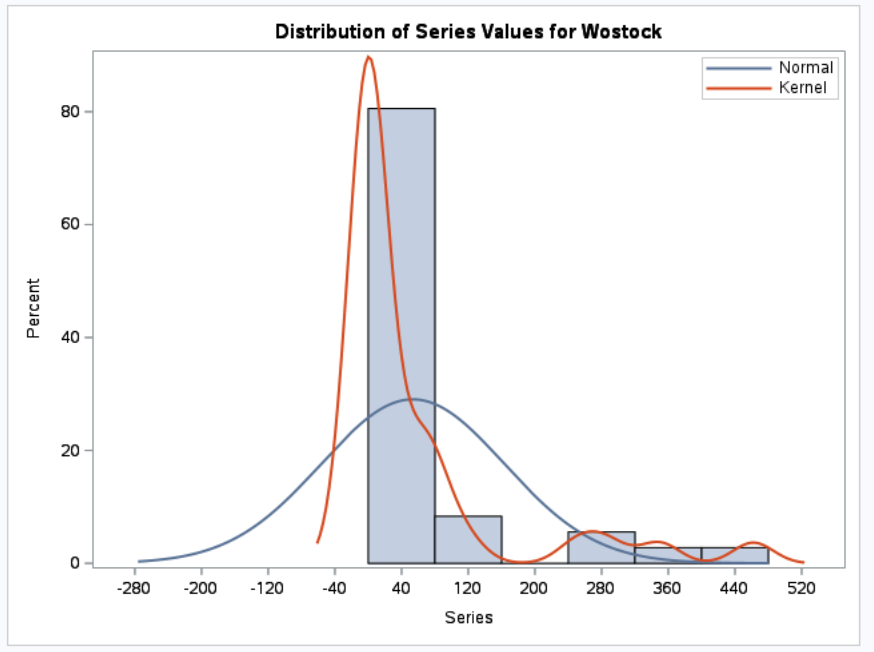
Tabulka 6 poskytuje statistický souhrn proměnné, která byla analyzována s použitím 36 pozorování. Průměrná hodnota proměnné je 22,812.5 s relativně nízkou standardní odchylkou 320.46, což naznačuje malou variabilitu v datech. Šikmost je velmi blízká nule, což ukazuje na symetrii distribuce, zatímco negativní špičatost naznačuje, že distribuce je méně špičatá než normální rozdělení. Koeficient variace je přibližně 1.4 %, což ukazuje, že standardní odchylka je malá ve srovnání s průměrem. Standardní chyba průměru je 53.41, což dává najevo přesnost odhadu průměru. Souhrn ukazuje konzistentnost a pevnou strukturu dat, která bude dále použita pro časovou řadovou analýzu.
Tabulka 7 obsahuje výsledky běžně používaných testů normálnosti na hodnocení předpokladu normálního rozdělení datové sady. Shapiro-Wilkův, Kolmogorov-Smirnovův, které jsem bral v úvahu, poskytují p-hodnoty, které jsou výrazně vyšší než 0.05, což indikuje, že neexistuje statisticky významný důvod zamítnout hypotézu normálnosti. Tento závěr naznačuje, že data mohou být považována za pocházející z normálního rozdělení.

Tests for Normality				
Test	Statistic		p Value	
Shapiro-Wilk	W	0.95652	Pr < W	0.1676
Kolmogorov-Smirnov	D	0.068606	Pr > D	>0.1500
Cramer-von Mises	W-Sq	0.051402	Pr > W-Sq	>0.2500
Anderson-Darling	A-Sq	0.384163	Pr > A-Sq	>0.2500

Tabulka 7 Testy normality pro doplněný set (vlastní zdroj)

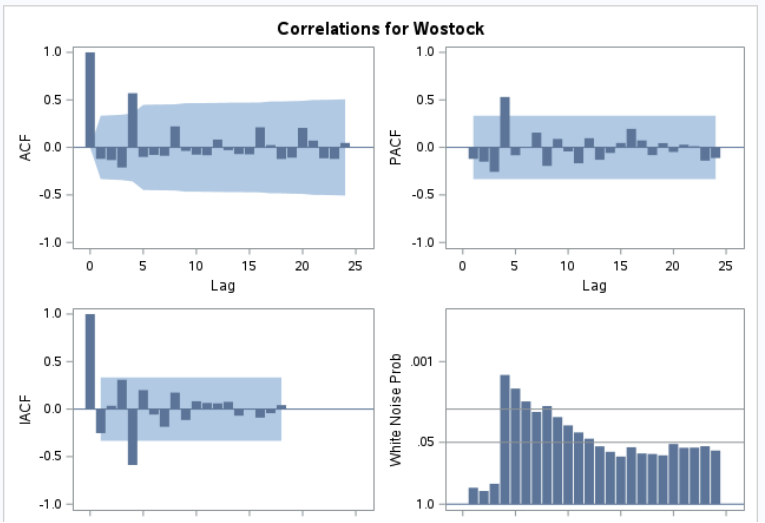
Na grafu 2 je histogram distribuce hodnot Wostock s překrývajícími se křivkami pro normální odhad a odhad hustoty pravděpodobnosti. Histogram ukazuje, že většina hodnot je

soustředěna kolem jednoho centrálního vrcholu, což naznačuje normální rozdělení. Křivka normálního rozdělení se nezdá dobře odpovídat datům, zatímco křivka odhadu hustoty pravděpodobnosti lépe odpovídá vrcholu distribuce.



Graf 2 Distribuce hodnot pro sérii – upravená data (vlastní zdroj)

Graf 3 zobrazuje grafy autokorelační funkce (ACF) a částečné autokorelační funkce (PACF). Jsou zobrazeny ve dvou horních panelech a obě ukazují, že autokorelace na většině zpožděních je minimální, což naznačuje slabou až žádnou lineární závislost mezi pozorováními v rámci série. Dolní grafy obsahují inverzní ACF, která není běžně používána, a histogram bílého šumu, který ukazuje distribuci autokorelací reziduí modelu. Relativně



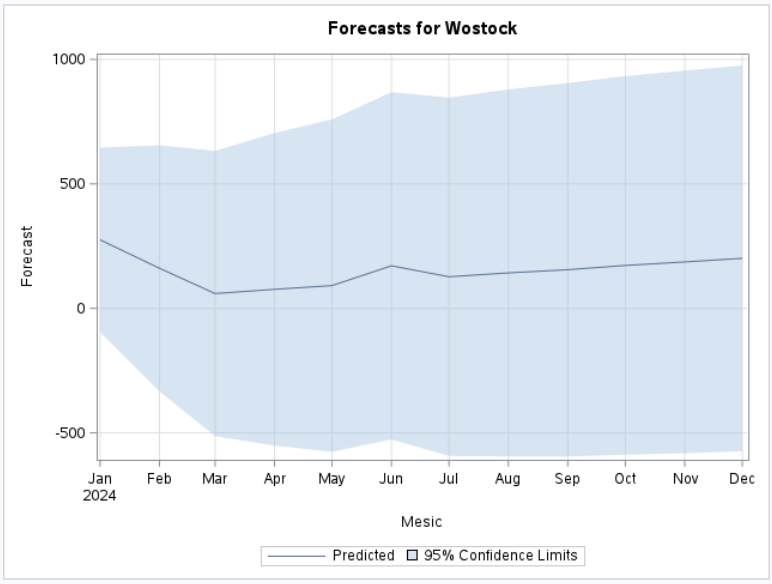
Graf 3 Korelační grafy pro sérii Wostok (vlastní zdroj)

rovnoměrné rozložení autokorelací v histogramu bílého šumu naznačuje, že rezidua modelu jsou dobře aproximována náhodnými fluktuacemi. Tyto grafy společně poskytují důkazy o tom, že předpoklady použitého časového řadového modelu jsou splněny.

4.5.8.2 Predikce časové řady

Model, který používám pro modelování predikce časové řady je ARIMA, což je autoregresní integrovaný klouzavý průměr. Tento model jsem zvolil po předchozí analýze upraveného setu dat.

Nastavil jsem hodnoty p=1, což značí použití jednoho předchozího pozorování pro autoregresní složku, d=1 což značí diferencování a jednu chybu q=1 pro klouzavý průměr. Dále jsem nastavil sezónní autoregresní řád na Q=1, sezonní řád diferencování D=1 a sezónní řád klouzavého průměru na Q=1. Perioda je nastavena na 12 měsíců.



Graf 4 Predikce upravené časové řady (vlastní zdroj)

Graf 4 poskytuje grafické znázornění predikovaných hodnot pro proměnnou Wostock v roce 2024. Předpovědi byly generovány pomocí vhodně specifikovaného časového řadového modelu a ukazují očekávané měsíční hodnoty s 95% intervaly spolehlivosti. Model předvídá sezónní variabilitu, se zvýšením hodnot v letních měsících a poklesem v období zimy, kdy je ke konci roku vidět vzestupná tendence. To může být ovlivněno chybějícími daty v souboru. Tato sezónní tendence je podložena dřívějšími pozorováními a analytickými předpoklady. Interval spolehlivosti, který je zobrazen jako světle modrá oblast, se rozšiřuje v časových okamžicích vzdálenějších od posledního pozorovaného bodu, což odráží

zvyšující se nejistotu v dlouhodobějších předpovědích. Celkově graf poskytuje vizuální reprezentaci očekávaných budoucích trendů a sezónních vzorců v proměnné Wostock.
Dá se tedy říct, že výsledek predikce není zcela jasný a je spíše orientační a na tomto základu jsem se rozhodl existující data do systému nezahrnovat, pouze se jimi inspirovat co se týče datové struktury.

5 Výsledky a diskuse

V této části diplomové práce shrnu výsledky vlastní práce, zaměřím se na podstatné části vývoje aplikace a poskytnu shrnutí a limity vývoje dané aplikace.

Podstatnou částí celé práce bylo analyzovat stávající manuální systém, který firma využívá k prodeji limonád svým zaměstnancům a navrhnout automatizované řešení. Tento systém má svoje omezení, která jsou očividná a přišel čas na aktualizaci procesů.

Systém navržený v této diplomové práci významně zjednodušuje procesy spojené s evidencí a správou zásob. Automatizace záznamů a vyúčtování snižuje riziko lidských chyb a výrazně tak šetří čas zaměstnancům i administrativnímu personálu poskytuje všechna data o prodeji v reálném čase.

Nedílnou součástí návrhu systému bylo stanovení technologií, ve kterých bude systém implementován. Zvolil jsem architekturu API, která je v dnešní době velice rozšířená a nabízí také možnosti pro budoucí rozšíření celého systému a využít tak obsah, které vystavuje back-endová část aplikace a je tak snadno použitelná pro ostatní aplikace, například pro mobilní vývoj, jelikož komunikuje pomocí typu JSON, který jsou schopny přijímat řady jazyků.

5.1 Zabezpečení

Stěžejní bod celého problému jsou technické a bezpečnostní aspekty. Zde je místo, kde může docházet k úniku dat, například při nezabezpečeném API nebo při vytváření tokenů či přístupových tokenů. Celá struktura API je privátní, takže pro komunikaci jsem využil JWT token, respektive Bearer autentizaci. Tento token je dobře šifrovaný a pravidelně obnovovaný, takže je velmi obtížné ho prolomit a celá aplikace běží ještě pod firemní VPN sítí, takže je zde velmi vysoká pravděpodobnost, že k prolomení tokenu nedojde, jelikož je zde použita dvojí ochrana proti proniknutí. Veškeré dotazy, které přichází z front-endové části obsahují specifický token podle uživatele, a tak jsou všechny dotazy zabezpečené. Nedílnou součásti autentizace je i autorizace. Zvolil jsem firemní službu KeyCloak, která obsahuje všechny zaměstnance z interního systému firmy. Díky tomu mohu efektivně vytvořit uživatele do databáze aplikace a evidovat jejich roli v rámci systému. Toto řešení je efektivní, a navíc splňuje integritu dat a redukuje jejich redundanci v rámci firmy, jelikož všechny data o zaměstnancích jsou evidována jen na jednom místě, a to ve firemním systému.

Administrátoři aplikace mají možnost, jak vytvořit objednávku, tak spravovat aplikaci, což vede k dalšímu vylepšení stávajícího systému, a to jsou přehledy o platbách a o dlužných částkách.

5.2 Tvorba systému

Celý systém byl vytvořený na základě specifikovaných požadavků, které jsem dal dohromady se členy back-office. Na tomto základě vznikla soupis požadavků, které bylo třeba naplnit pro efektivní a funkční nástroj, který usnadní čas a peníze firmě. Jelikož se jedná o vývoj systému na zakázku, tak jeho naprogramování a realizace trvala delší dobu, kdyby se zvolil již existující systém. Tento systém na zakázku však přinese několik výhod, jako je například jeho dlouhodobá udržitelnost, správa ostatními programátory a jednoduchou integraci s již existujícími systémy. Na druhou stranu bude delší dobu trvat odladění chyb, otestování celého systému a než se program stane plně funkční a schopný nasazení do plného produkčního prostředí. Dalším aspektem odladění aplikace je reakce na chyby spojené s integrací interních nástrojů firmy, ale tato akce by se týkala i již existujícího řešení a bylo by například nutné do jisté míry prostředí upravit pro použití těchto interních firemních aplikací.

5.3 Modul skladu

Modul skladu je vytvořen na základě analýzy existujících systému a je upraven pouze pro potřeby daného systému. Administrátoři mohou naskladňovat produkty, mohou produkty odepsat anebo přesunout na danou lokalitu. Všechny sklady a lokality si mohou vytvořit a distribuovat produkty dle potřeby. O všech přesunech či odpisech jsou vedeny záznamy, takže mohou bezpečně dohledat jakýkoli přesun či změnu ve skladu nebo lokalitě. Oproti existujícím systémům je tento zjednodušený a nabízí řadu vylepšení, které by se mohli implementovat, ale dle specifikací požadavků zatím nebyl vysloven žádný podnět k vylepšení.

5.4 Analýza existujících dat

Analýza existujících dat ukázala, že daná data jsou nevhodná pro integraci do systému, jelikož neobsahují všechny hodnoty a zkreslovala by tím statistické ukazatele v systému. Pomocí statistických nástrojů byla doplněna o hodnoty, ale ani tak se neukázalo, že by byla

vhodná, a tak integrace do existujícího systému nebude možná. Systém tak bude pracovat až s hodnotami, které se do systému budou propisovat v rámci používání aplikace.

V této práci jsem vytvořil systém, který je ve verzi 1.0.0. Už ze samotného návrhu a specifikace požadavků vyplynulo několik skutečností, které by bylo dobré zahrnout do dalšího vydání aplikace. Jedná se například o vytvoření mobilní aplikace, která bude lépe vyhovovat nativním specifikacím telefonů, implementace platební brány, propojení plateb s bankou a aplikací nebo implementace hardware, který bude komunikovat s aplikací a otevře lokalitu až po přihlášení a naskenování kódu.

6 Závěr

Na závěr této diplomové práce lze konstatovat, že navrhovaný systém představuje funkčně a datově krok vpřed ve zlepšení procesů správy skladového modulu a konkrétního prodeje limonád a dalších produktů ve firmě. Aktuální manuální systém je neefektivní, náchylný k chybám a časově náročný jak pro zaměstnance, tak pro BackOffice. Nově navrhovaný systém, založený na PostgresSQL databázi, Symfony frameworku pro back-end část a Vue.js pro část front-endu, nabízí řešení těchto problémů.

Systém poskytuje automatizaci mnoha procesů, včetně sledování zásob, jejich koordinaci v rámci jednotlivých lokalit, objednávání a vyúčtování. Umožňuje také reálný čas sledování a řízení zásob limonád, což zvyšuje efektivitu a snižuje možnost vzniku chyb. Integrace s mobilními zařízeními a jednoduché uživatelské rozhraní, vytvořené pomocí Vue.js, zajistí snadný přístup pro všechny zaměstnance. Systém klade velký důraz na bezpečnost, kdy je využita firemní VPN a dále zabezpečené API spolu se službou KeyCloak.

Navíc, díky digitalizaci a centralizaci dat, systém nabízí lepší přehled o finančních operacích a pomáhá snižovat dlouhodobé dluhy zaměstnanců. Také usnadňuje administrativní práci, jelikož eliminuje potřebu manuálního záznamu a zpracování dat.

Výzvou zůstává zajištění plynulé implementace systému a přizpůsobení se změnám ze strany zaměstnanců. Finanční náklady a potenciální technické problémy jsou také důležitými aspekty, které je třeba zvážit. Přesto celkově systém slibuje výrazné zlepšení efektivity, transparentnosti a uživatelského komfortu při správě prodeje limonád a ostatních produktů ve firmě.

V porovnání s implementací již existujících systému je tato cesta časově a dá se říct i finančně náročnější než aplikace existujícího řešení, ale přináší několik výhod, které jsou pro firmu důležité, jako je snadná udržitelnost a přehled o změnách v aplikaci, zpětná kompatibilita a reakce na problémy spojené s integrací firemních služeb, které aplikace využívá.

Z výsledků analýzy stávajících dat bylo poukázáno na nedostatečně vedené záznamy a tím pádem větší náchylnost k chybám, které mohou vést k ovlivnění stavu produktů či platebních závazků. Dále bylo zjištěno, že se zcela jasně nedá predikovat objem konkrétního produktu v čase vzhledem k těmto chybám v datech, a tak není dobré tyto data zahrnovat do přehledů ve vzniklém systému.

Přínos této práce tkví v automatizaci, digitalizaci a ušetření zdrojů firmě při jejím prodeji a správě produktů v reálném čase. Eliminuje chyby a přináší nový pohled na data získaná prodejem pro další analyzování a výsledné závěry směrem k hospodaření s produkty. Práce může sloužit také jako podklad pro kolegy, kteří by chtěli nad systém provádět další úpravy nebo vylepšení.

7 Seznam použitých zdrojů

1. ADEPOJU, Fikayo, 2020. Introduction to GraphQL. CIRCLE INTERNET SERVICES, INC. *Cicrleci Blog* [online]. [cit. 2023-11-28]. Dostupné z: https://circleci.com/blog/introduction-to-graphql/

2. ARCHON SYSTEMS INC, 2023. Software pricing. ARCHON SYSTEMS INC. *InFlow* [online]. [cit. 2023-11-28]. Dostupné z: https://www.inflowinventory.com/software-pricing

3. BOOTSTRAP TEAM, [2023]. Get started with Bootstrap. BOOTSTRAP TEAM. *Bootstrap* [online]. [cit. 2023-11-28]. Dostupné z: https://getbootstrap.com/docs/5.3/getting-started/introduction/

4. BRUCKNER, Tomáš, 2012. *Tvorba informačních systémů : principy, metodiky, architektury*. 1. Praha: Grada. ISBN 978-80-247-4153-6.

5. COMPOSER, 2023. Getting Started. COMPOSER. *Composer* [online]. [cit. 2023-11-28]. Dostupné z: https://getcomposer.org/doc/00-intro.md

6. DOCKER INC., c2013–2023. Docker overview. DOCKER INC. *Docker* [online]. [cit. 2023-11-28]. Dostupné z: https://docs.docker.com/get-started/overview/

7. DOCTRINE, [2023]. Getting Started with Doctrine. DOCTRINE. *Doctrine* [online]. [cit. 2023-11-28]. Dostupné z: https://www.doctrine-project.org/projects/doctrine-orm/en/2.17/tutorials/getting-started.html

8. DUNGLAS, Kévin, c2023. API Platform's Philosophy. *API Platform* [online]. [cit. 2023-11-28]. Dostupné z: https://api-platform.com/docs/extra/philosophy/

9. Getting Started, [2022]. *Axios* [online]. [cit. 2023-11-28]. Dostupné z: https://axios-http.com/docs/intro

10. IBM, 2023. What is a REST API? IBM. *IBM* [online]. [cit. 2023-11-28]. Dostupné z: https://www.ibm.com/topics/rest-apis

11. INFLUXDATA INC., c2023. What is time series data? INFLUXDATA INC. *InfluxData* [online]. [cit. 2023-11-29]. Dostupné z: https://www.influxdata.com/what-is-time-series-data/

12. JUBA, Salahaldin a Andrey VOLKOV, 2019. *Learning PostgreSQL 11* [online]. 3. Packt Publishing, Limited [cit. 2023-11-28]. ISBN 9781789535211. Dostupné z: https://ebookcentral.proquest.com/lib/techlib-ebooks/reader.action?docID=5683528

13. KARRYS, Luke, Michael RIENSTRA, Myles BORINS a Edward THOMSON, 2023. About npm. *Npm Docs* [online]. [cit. 2023-11-28]. Dostupné z: https://docs.npmjs.com/about-npm

14. MALIK, Nyma, 2017. The Difference Between REST and SOAP APIs. DZONE. *DZone* [online]. [cit. 2023-11-28]. Dostupné z: https://dzone.com/articles/difference-between-rest-and-soap-api

15. SEDLÁČEK, Petr, 2023. Itnetwork.cz. ITNETWORK.CZ. *IT Network* [online]. [cit. 2023-11-28]. Dostupné z: https://www.itnetwork.cz/javascript/nodejs/rest-api-soap-graph-a-json

16. SORTLY INC, 2023. Inventory management. SORTLY INC. *Sortly* [online]. [cit. 2023-11-28]. Dostupné z: https://www.sortly.com/solutions/inventory-management-software/

17. SORTLY INC, 2023. Pricing. SORTLY INC. *Sortly* [online]. [cit. 2023-11-28]. Dostupné z: https://www.sortly.com/pricing/

18. SYMFONY™, [2023]. Installing & Setting up the Symfony Framework. SYMFONY™. *Symfony* [online]. [cit. 2023-02-13]. Dostupné z: https://symfony.com/doc/current/setup.html

19. TABLEAU SOFTWARE, c2003–2023. Time Series Analysis: Definition, Types, Techniques, and When It's Used. TABLEAU SOFTWARE. *Tableau* [online]. [cit. 2023-11-29]. Dostupné z: https://www.tableau.com/learn/articles/time-series-analysis

20. THE GRAPHQL FOUNDATION, 2023. GraphQL. THE GRAPHQL FOUNDATION. *GraphQL* [online]. [cit. 2023-11-28]. Dostupné z: https://graphql.org/

21. THE PHP GROUP, c2001-2023. History of PHP. THE PHP GROUP. *Php* [online]. [cit. 2023-11-28]. Dostupné z: https://www.php.net/manual/en/history.php.php

22. THE SASS TEAM, c2006–2023. Documentation. THE SASS TEAM. *Sass* [online]. [cit. 2023-11-29]. Dostupné z: https://sass-lang.com/documentation/

23. VUE-CHARTS TEAM, [2016]. Getting Started. VUE-CHARTS TEAM. *Vue-chartjs* [online]. 15.11. 2023 [cit. 2023-11-29]. Dostupné z: https://vue-chartjs.org/guide/

24. YOU, Evan, c2014–2023. Introduction. *Vue.js* [online]. [cit. 2023-11-28]. Dostupné z: https://vuejs.org/guide/introduction.html

25. ZWASS, Vladimir, 2023. Information system. *Britanica* [online]. 10.11. 2023 [cit. 2023-11-28]. Dostupné z: https://www.britannica.com/topic/information-system

8 Seznam obrázků, tabulek, grafů a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam grafů