Abstrakt

text

Summary

text

Klíčová slova

text

Key words

text

Bibliografická citace

text

Čestné prohlášení

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů jsou úplné a že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne xx. x. 2017

.....

jméno

Poděkování

text

Obsah

ÚVOD

tex

CÍL DIPLOMOVÉ PRÁCE

Hlavním cílem této práce je vytvořit platformu, která bude schopná zpracovat data zadaná uživatelem, analyzovat je a na základě vnitřní logiky a informace nesené v těchto datech je uložit do logických celků. Tyto celky poté zobrazit uživateli, nechat jej dále definovat dodatečné informace, provádět reporting případně zobrazit v přehledných grafech.

Platforma se bude zaměřovat převážně na uživatelské rozhraní, tak aby její používání bylo co nejintuitivnější a nejjednodušší.

METODIKA PRÁCE

Metody

Nejdůležitějším zaměřením této platformy je uživatelská přívětivost a jednoduchost na používání, proto bude při vývoji kladen důraz na spokojenost uživatelů. Tohoto bude dosaženo použitím agilních metodik při vývoji, kdy bude postupně dodáváný produkt předáván úzkému kruhu uživatelů, kteří se budou vyjadřovat k uživatelskému rozhraní. Platforma bude psána jako webová aplikace, která bude přistupovat do databáze přes rozhraní napsané v jazyce Java.

V části zpracování dat bude použito několik ETL metodik a data mining technik, které povedou k získání logických informací ze zadaných informací. Platforma bude vyvýjena s možností škálovatelnosti a použití nad velkým objemem dat.

Postupy

K vytvoření co nejpřívětivější platformy budou využity zkušenosti a knižní publikace zabývající se tímto tématem. Dále budou analyzovány jednotlivé postupy zadávání dat uživatelů do takovéhoto systému, které povedou ke zpřehlednění a zjednodušení používání.

Pro komunikaci se serverem bude použit standard REST, kterýusnadní komunikaci se serverem a umožní případné navázání nových aplikací. V případě že bude vytvořena mobilní aplikace pro záskávání dat nebude nutné psát znovu stejnou nebo podobnou logiku.

Pro zabezpečené přihlášení do aplikace bude použit autentikační server, který bude zajišťovat výtváření a správu uživatelů spolu s jejich právy.

1 TEORETICKÁ VÝCHODISKA

Pro plné pochopení, výběru a případném vypracování platformz je potřeba si objasnit a vysvětlit několik témat. Jsou to především *Vývojové platformy low-code*, výsledná platforma by měla splňovat tuto definici. Dále si objasníme pojemy *Bussiness intelligence* (platforma bude z části pracovat s touto oblastí) a *Platforma pro pokročilou vizualizaci dat* – pro snadné používání uživatelského rozhraní. A vzhledem k tomu že výsledná platforma musí do určité části pracovat s uživatelskými právy a spravovat uživatele, objasníme si pojem *Server pro řízení přístupu a identity*

1.1 Vývojové platformy low-code

Vývojové platformy low-code jsou celkem nový pojem, tyto produkty začali vznikat, protože malé a střední podniky potřebovali vytvořit rychle a za použití menšího počtu vývojářů aplikace, které mohou být nadále rychle spravovány. [1]

Toto v podstatě znamená, že vývojáři mohou rychle měnit software na základě uživatelských požadavků, což má za následek spokojenější uživatele, uživatelsky přívětivější software a toto všechno za minimálního použití ručního programování. Takovéto platformy neeliminují programování jako takové, ale napomáhají rychlejšímu vývoji, tak že poskytují vizuální nástroje a napomáhají konfiguraci datových modulů a pomáhají eliminovat problémy spojené s datovou integrací. [2]

Výhody low-code platforem

- **Produktivita:** Systémy mohou být vyvýjeny a nasazeny během menšího časového rozmezí, oproti klasickému programování. [3]
- **Reakční schopnost:** Vývojář může často zvolit různé druhy platforem na kterých bude výsledný produkt fungvat, od mobilních aplikací, až po webové služby. [3]
- **Spolehlivost:** Aplikace mohou být aktualizovány mnohem rychleji, což má za následek jejich stabilitu a spolehlivost. [3]
- **Úspora času a peněz:** Vývojáři mohou vytvořit mnohem více funkcionality za kratší čas, z čehož plyne že si firma může dovolit mensí počet programátorů. [3]

- **Zaměření na samotný vývoj:** Zaměřením na to co má aplikace dělat, a ne jak to má dělat, programátoři se mohou zaměřit na funkcionalitu a uživatelskou spokojenost. Při vývoji je možné se zaměřit také více na uživatelské požadavky mnohem rychleji. [3]

1.1.1 Příklady low-code platforem

Microsoft PowerApps Vývojová platforma od firmy Microsoft, která dovoluje vytvořit během několika málo kliknutí aplikaci pro mobilní platformy a také jako webové služby. Při spojením této platformy a aplikace Power BI vzniká velice robustní vývojářský nástroj, díky kterému je možné rychle integrovat produkční data do aplikace, kterou budou uživatelé rádi používat. [1]

Zoho Creator Výhodou této platformy je využití techniky „`drag-and-drop`", která umožňuje vytvářet aplikace a převážně jejich uživatelské rozhraní bez nutnosti psát jakýkoliv kód. [4]

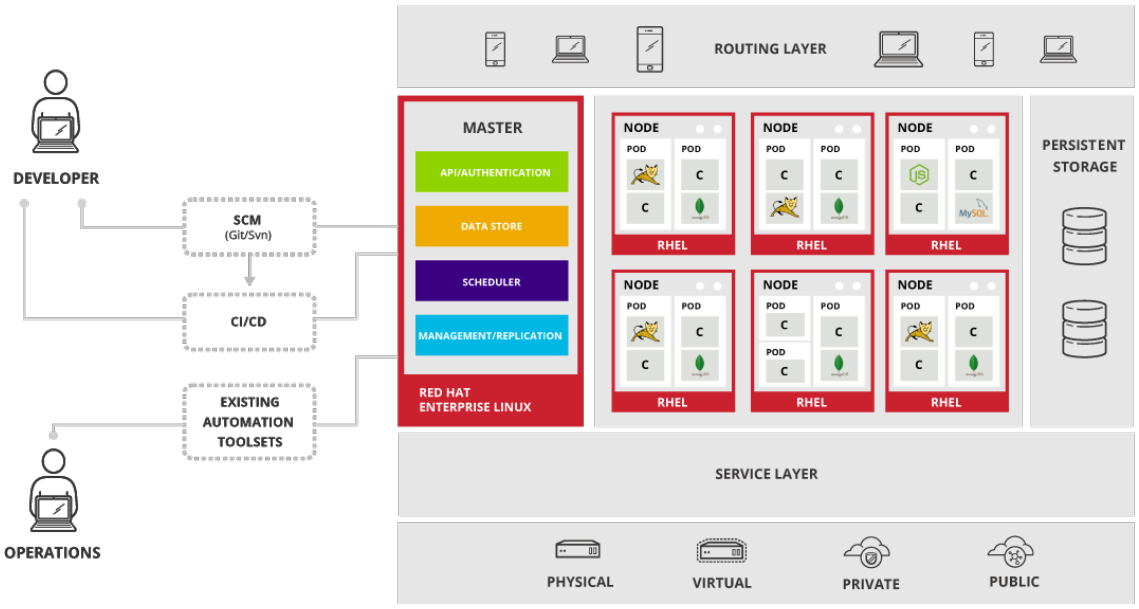
Rollbase Při používání této platformy vývojář jako první definuje objekty, jejich vlastnosti a vztahy mezi těmito objekty. Po překonání tohoto kroku máme již plně funkční webovou aplikaci, která je funkční napříč všemi mobilními zařízeními. [5]

Openshift Platforma pro vývoj webovým a mobilních aplikací, postavená na kontejnerech, které zajišťuí rychlý vývoj a možnost dedikovat vývojáře na vytvoření jednoduchých funcionalit jako samostatné aplikace [1], které za pomocí Openshiftu vytvoří velkou a komplexní aplikaci. Na Obrázku **??** můžeme vidět z jakých vrstev se skládá Openshift a můžeme vidět jednotlivé aplikace zázorněné v nodech, dále můžeme vidět jaké nástroje nabízí Openshift vývojářům (management zdrojových kódů – SCM a kontonualní integrace – CI/CD).[6]

1.1.2 Cloudový výpočet

Výpočet pomocí cloudu je další fáze ve vývoji internetu, cloud v tomto použití znamená že všechno potřebné pro vývoj a hostování aplikací, až po samostatné stroje je možné nabídnout jako služba kekoliv na světě se člověk nachází. I když

[1] Takovýmto aplikacím se říka Microservice `https://smartbear.com/learn/api-design/what-are-microservices/`



Obrázek 1.1: Znozornění jednotlivých vrstev v platformě openshift.

poskytovatelé cloudových řešení mají často velice robustní bezpečnostní systém je na uživateli, který má uložené data v cloudu, aby zajistil jejich bezpečnost. To znamená že pokud data uniknou z cloudu díky špatnému zabezpečení v aplikaci, která je hostovaná, není chyba poskytovatele, ale firmy, která takovou aplikaci vydala.[7]

Pporovnání ceny cloudového výpočtu a klasického datavého uložiště není až tak jednoduché, záleží na někkolika faktorech. Vezměme si například lokaci datového uložiště, pokud například cena elektické energie v místě datového uložiště je velice levná firma nemusí být tolik tlačena do cloudového řešení. ale pokud k těmto datům přistupuje velké množství uživatelů z různých koutů světa může se stát že námi poskytované služby budou neresponzivní a uživatelé mohou odejít ke konkurenci. V tomto momentě je potřeba zvážit zda se nám cloudové řešení vyplatí a kdy ne. Důležité je také uvědomit si, že 42 % nákladů na datové uložiště jde do hardware a software (tyto náklady jsou rozloženy v průběhu času) a 58 % nákladů jde do topení, klimatizace, daní a samostatné práce. [7]

Typy cloudového výpočtu

- **Veřejné** dostupné pro širokou veřejnost, jak zdarma, tak placené verze.
- **Soukromé** často používané firmami skupinami uživatelů, kteří potřebují zabezpečit data. Často velice drahé a ačasově nákladné řešení.

- **Komunitní** podobné soukromým, ale rozšířené mezi větší skupiny lidí.
- **Hybridní** vytvořené z jednoho a více druhů, privatního a nebo veřejného cloudu. Mezi jejich portfoliočastopatří zálohakekritickýmslužbám.
- **DaaS** data jako služba, pouze data uložené v cloudu.
- **PaaS** pro vývoj a hostování celého vývojového cyklu, často všetně možnosti nasazení výsledné aplikace.
- **IaaS** infrastruktura jako služba, opravdové, nebo virtuální počítače nabízené uživatelům. [8]

Platforma jako služba – PaaS

Platforma jako služba tento pojem označuje službu, která zahrnuje kompletní škálu nástrojů sloužících pro vývoj aplikací. Od databází, přes aplikační rámce a testovací nástroje až po nasazení a překlad aplikace. Výhoda této služby je převážne v tom, že všechno je přístupné přes internet a často jako webová aplikace, takže není nutné kupovat často velice drahé nástroje. Někteří poskytovatelé nabízí také vyrovnání zatížení, to znamená že pokud je výsledná aplikace pod vysokým náporem uživatelů, automaticky se přiřadí prostředky, aby uživatelé nezaznamenali pád aplikace a bez nutnosti zasáhnout do nastavení služby.[9]

Výhody PaaS

- **Méně kódu** – Díky možnosti projení několika menších aplikací dohromady není potřeba psát stejný kus kódu pořád dokola.
- **Nové možnosti bez potřeby nabírání nových lidí** – Díky PaaS dostane tým do rukou sofistikovanější nástroje.
- **Vývoj pro více platform** – Výhodou mnoha poskytovalů služby PaaS je možnost překladu aplikací pro různé platformy (několik mobilních platforem a webová aplikace).
- **Propojení geograficky nesourodých týmů** – Pokud je tým rozdělen po různých částech světa služby PaaS dovolují takto rozděleným týmům pracovat efektivněji.
- **Efektivní životní cyklus aplikace** – V rámci integrovaného prostředí se často nachází funkce pro podporu životního cyklu aplikace (sestavení, nasazení,

otestování, správa, aktualizace...). [10]

1.2 Bussiness Intelligence

Nástroje pro bussiness Intelligence zahrnují jak samostatná data, tak časovou jednotku, takže můžeme nad těmito daty provádět predikci pomocí sofistikovaných nástrojů a výpočtů. Ze začátku byo jednoduché provádět takové výpočty, protože jednoduše firma nesbírala takové množství dat. Aktuálně však není v lidských silách provádět takové výpočty nad tak obrovským množstvím dat. [11]

Pravděpodobně nejvíce rozšířeným aplikováním data-miningu je marketing – sledování nakupování a chování zákazníků. V této oblasti je možné vybrat si každého zákazníka, a v případě že máme dostatek dat, cílit na něj lépe reklamu. [11]

Pro plné využití BI nástrojů potřebujeme sledované subjekty rozdělit do několika skupin, tyto skupiny musí být co nejvíce **Heterogenní** (rozdílné) vůči sobě, a subjekty v rámci jedné skupiny musí být na druhou stranu co nejvíc **Homogenní** (stejné). Pokud sledované subjekty spadají do více skupin (toto se může hodit z mnoha důvodů) můžeme použít takzvané **Clustery**. Což jsou skupiny objektů, které si jsou co nejvíce podobné, ale objekty v jiné skupině se jim budou co nejvíce lišit. [11]

Samostatá data jako taková však nejsou to nejdůležitější, pokud nebudeme schopni vhodně interpretovat a využít data, která jsme nashromáždili jsou nám k ničemu, z toho důvodu vzniklo strojové učení. V podstatě to znamená, že pokud předáme stroji dostatečně velké možství dat a nastavíme správně parametry, tak nám mohou stroje umožnit rychlejší interpretaci takových dat. Ale na předpovědi je nutno nahlížet s odstupem a nebrat je příliš vážně a přesně, naštěstí například pro úspěšný prodej není potřeba přesných předpovědí, stačí pouze vědět kdy a komu poslat přesně cílenou reklamu. Pokud se systém trefí do těhcot kritérií je velká pravděpodobnost že si námi zacílený zákazník pořídí produkt, který se mu snažíme prodat. [12]

1.2.1 Big data

Označení Big data může dostat jakékoliv množství strukturovaných, nestrukturovaných a částečně strukturovaných dat, které mají poteciál k tomu aby z nich bylo

možné vydolovat nějaké zkryté informace. Ve zkratce to znamená že data začnou výt velkými v momentě, kdy jejich zpracování tradičními metodami je časově a technologicky složité.[13]

Big data lze definovat pomocí pravidla **3V** – Volume, Velocity, Variety. Pro přesné definování můžeme vzít v úvahu také Veracity, Validity a Volatility.[13]

- **Volume** – Ve světě Big data máme na mysli opravdu velké možství dat.
- **Velocity** – Rychlost s jakou jsou data ypracována.
- **Variety** – Různorodostí dat máme na mysli jejich formát (*strukturovaná* – klasické RDBMS, *částečně strukturovaná* – emaily, zprávy ...; *nestrukturovaná* – multimediální obsah).
- *Veracity* – Věrehodostí rozumíme že data musí být očištěna od zbytečného šumu.
- *Validity* – Data musí být co nejpřesnější a co možnáa nejvhodnější pro naše rozhodování.
- *Volatility* – Data musí být co nejaktuálnější pro přesnější predikci a jejich pozdější zpracování. [13]

Nejvhodnější místo pro uložení takového množství dat, je cloud, přesněji využít službu některých poskytovatelů privátních nebo veřejných IaaS. 1.1.2 Cloud je vhodný pro Big data převážně kvůli tomu, že jak uložení, tak práce s takovým objemem dat, požaduje velké množství distrubuované počítačové síly. [14]

Škálovatelnost se zaměřením na Hardware znamená že pro Big data je potřeba přejít z relativně malého výpočetního výkonu na velký během několika chvil bez nutnosti změnit architekturu. Pokud budeme mluvit o Softwaru je zapotřebí zachovat stejnou jednotku síly jak se zvětšuje Hardware (při zvýšení objemu dat by mohlo dojít ke značnéu poklesu výkonu, pokud na to není systém připraven). [14]

Pružnost je potřeba zachovat co největší, převážně kvůli tomu že mohou nastat momenty, kdy máme velké množství dat, které se ale v čase mohou smrsknout pouze na zlomek těchto dat. Pokud se tomu tak stane nechceme nadále platit za zbytečně nevyužitý prostor. [14]

Sdružování prostředků cloud dovoluje vytvářet skupiny sdílených prostředků. [14]

Samoobsluha většina poskytovatelů clouhových řešení nabízí možnost samostatně, bez nutnosti kontaktovat IT oddělení, navýšit zdroje, případně je odpojit (pokud již nejsou potřebné). Toto je často řešeno nějakým portálem, prípadně specifickými nástroji u uživatele daného cloudu. [14]

Nízká pořizovací cena znamená že často cloudové řešení nestojí uživatele tolik jako pořízení drahých datových skladů. [14]

Platba za chodu se používá u poskytovatelů cloudu jako způsob platby za zdroje, které využíváme. Takže je dost možné že v průběhu používání cloudu budeme platit různé částky. [14]

Tolerace pádů u cloudových řešení je nutnost a musí být u takovýchto řešení co nejnižší, aby byl zajištěn nepřetržitý chod. [14]

1.2.2 Data-mining

Pokud budeme potřebovat velkémožstvídat (Big data 1.2.1), musíme je nejdříve očistit od chybně zapsaných dat. Dokonce i když máme data ve standrardní formě nemůžeme počítat s tím, že jsou tyto data naprosto bez chybná. Takto chybně zapsaná data mohou být nicméně důležitá pro náš systém.[15]

Dále některé atributy mohou naprosto chybět pro některé záznamy, pro vypořádání s takto chybějícími atributmi můžeme zvolit jednu ze dvou taktik. [15]

1. Odstranění – jednoduše odstraníme celý záznam ve kterém se nám nachází nějaký chybějící atribut.
2. Náhrada – použijeme nejvíce frekventovanou, nebo nějakou výchozí hodnotu na místě ve kterém najdeme chybějící atributy. [15]

Na obrázku **??** vidíme jednotlivé fáze popsané v CRISP-DM [2] modelu, tyto fáze jsou:

Pochopení podniku na začátek je potřeba pochopit, co za problém se snažíme vyřešit pomocí dolování dat. Tento první krok je nejdůležitější, ze začátku budeme

[2] Zkratka znamená cross-industry process for data mining

mít pouze slabé pochopení tohoto kroku a v rámci procesu dolování dat se budeme vracet do tohoto bodu. [16]

Pochopení dat často narazíme na data, která nebudou naprosto sedět zadanému problému, proto se musíme zaměřit na jejich silné a slabé stránky. Často se stává že historická data neodpovídají aktuálním problémům podniku. Další složkou, která se projeví do dat je samozřejmě cena, některé datové sady jsou takřka zadarmo, další se dají pořídit a některá prostě neexistují. Proto v rámci pochopení dat musíme zvážit případné rozšíření datové složky a zda se nám vyplatí investovat do dalších dat. [16]

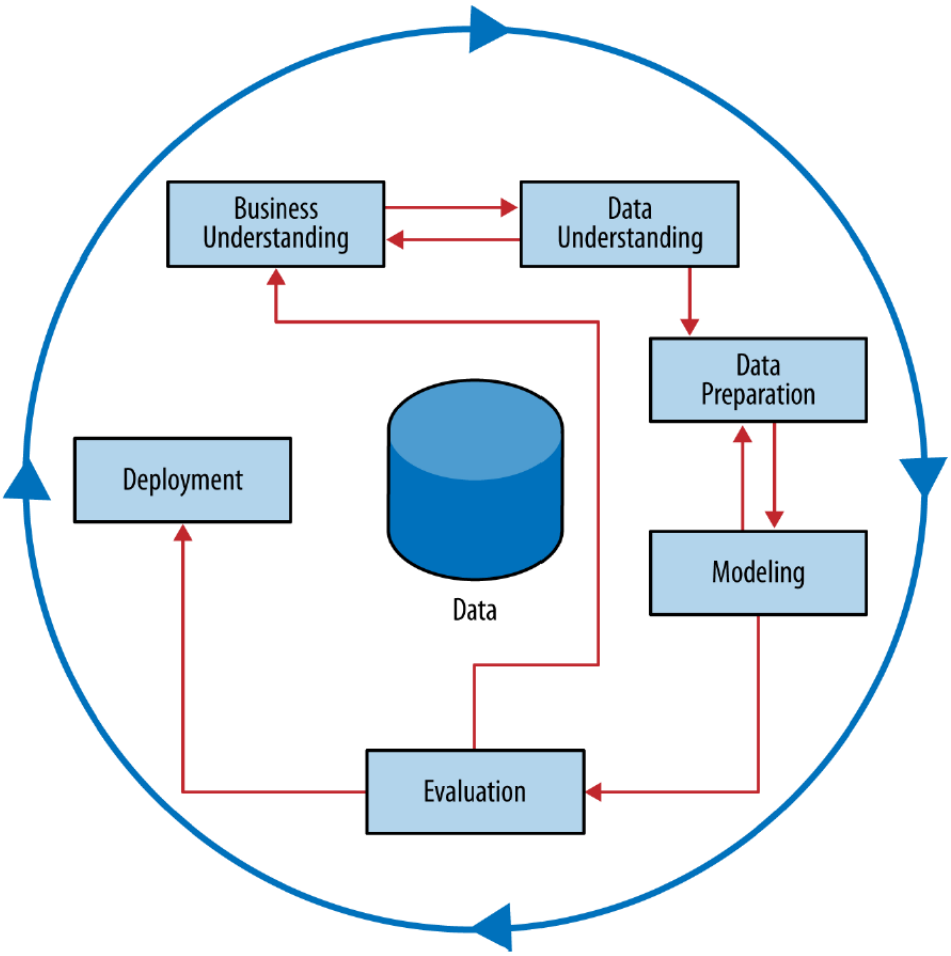
Příprava dat je dost možné, že tato část bude předcházet **pochopení dat**, protože jednoduše pro pochopení dat musíme nejdříve tyto data očistit od případných chybných a chybějících atributů. [16]

Modelování primární fáze ve které jsou použity techniky pro dolování dat. [16]

Vyhodnocení po aplikování modelování se nacházíme ve fázi, kde musíme ověřit, zda výsledný model je to co podnik potřebuje. Pokud tomu tak není, vracíme se zpátky na pochopení podniku a opakujeme celé kolečko znovu. Pro vyhodnocování používáme jak kvantitativní tak kvalitativní kritéria. [16]

Nasazení po vyhodnocení že námi zvolený model je v pořádku a že techniky zvolené pro dolování dat byly korektní aplikujeme výsledek na reálné využití. Bez ohledu na to zda nasazení proběhlo v pořádku vracíme se zpátky do první fáze – pochopení podniku. Další proces dolování dat může vyvodit jiný model, který může být vhodnější pro daný problém. [16]

Extraction, Transofrmation, Loading Extraction, Transofrmation, Loading ve zkratce **ETL** jsou procesy zodpovědné za naplnění dat do datových skladů. Nejdříve jsou data vyextrahována z různých uložišť. Těmito uložišti mohou být OLTP, starší systémy, webové stránky atd. Poté jsou data očištěna, zbavena duplikátů a chybějících atributů. Následně jsou data nahrána do datových skladů, kde se s nimi může dále pracovat.



Obrázek 1.2: Fáze CRISP-DM, referenční model.

1.3 Platforma pro pokročilou vizualizaci dat

Podniky často zjišťují že vizualizace dat je pro ně důležitým prvkem v odhalování problémů a monitorování podniku.Často se stává že klasické zobrazení dat v podobně reportů za použití tabulek nedokáže zaměřit celý problém, nebo často vede ke špatné analýze. Z toho plyne že je lepší použití grafických prvků jako (jednoduše řečeno grafů), pro ještě lepší usnadnění analýzy je vhodné použít dynamických a interaktivních grafických prvků. Mezi tyto dynamické a interaktivní prvky patří dashboardy, grafy a tabulky které se automaticky aktualizují pokud se jejich datová sada změní.

Pokročilá vizualizace dat vs. statické grafy

- **Dynamické datové sady** – Při změně datových sad (databází) se automaticky graf překreslí.
- **Vizuální dotazování** –Jednoduchá manipulace s grafy, to znamená například kliknutím na sloupec se provede akce, která má za následek překreslení grafu.
- **Několik dimenzí, propojené vizualizace** – Klasický graf nedokáže zobrazit závislosti mezi několika dimenzemi, proto je vhodné zobrazit provázané grafy, které reagují v závislosti na navigaci v jedné dimenzi. Například – počet prodaných kusů v čase, při výběru specifického měsíce a roku se automaticky překreslí graf na počet prodaných kusů za den.
- **Animované vizualizace** – Pokud má dimenze velké množství hodnot je vhodné použít animaci pro jednoduché ovládání a znázornění.
- **Zosobnění** – Analytici mohou mít různé pochopení dat a proto je dobré je nechat pracovat individuálně s daty. Často se také může stávat že různí analytici mají přístup k různým daotvým sadám, proto je potřeba zajistit pověřovací úrovně.
- **Varování pro podnik** – Pokud se stane že na obrazovce se nachází příliš mnoho dat může se často stávat že analytici jednoduše nebudou schopni odhalit problém včas. Proto by platforma pro pokročilou vizualizaci dat měla nabízet nějaký druh upozorňování. Tato upozorňování však nesmí být pouze grafického typu, ale také ve formě nějaké zprávy v případě že se uživatel nedívá přímo na vizualizaci. [17]

1.3.1 Single page aplikace

Tradiční přístup k vývoji webové aplikace je že máme databázi připojenou do serverového back-endu ke kterému jsme schopní se dostat pomocí webového rozhraní. Celá těžká práce je založena na serveru, to znamená navigace, veškerá logika a práva. To často vede k zatížení serveru a má za následek neresponzivní a pomalé aplikace. Z toho důvodu byl vyvinut nový způsob zobrazování aplikace, takzvané **Single page aplikace** (zkráceně pouze SPA). Tyto aplikace využívají JavaScript k vykreslení stránky a navigaci, to znamená méně dotazů na server při přechodu

mezi jednotlivými částmi aplikace. V případě že server budeme využívat pouze jako zdroj informací, je možné uživateli doručit intuitivní a snadno použitelnou aplikaci, kdy například ani pád serveru, nefunkční nebo pomalé připojení, nemusí znamenat pád aplikace [3]. [18]

Výhody Single page aplikací

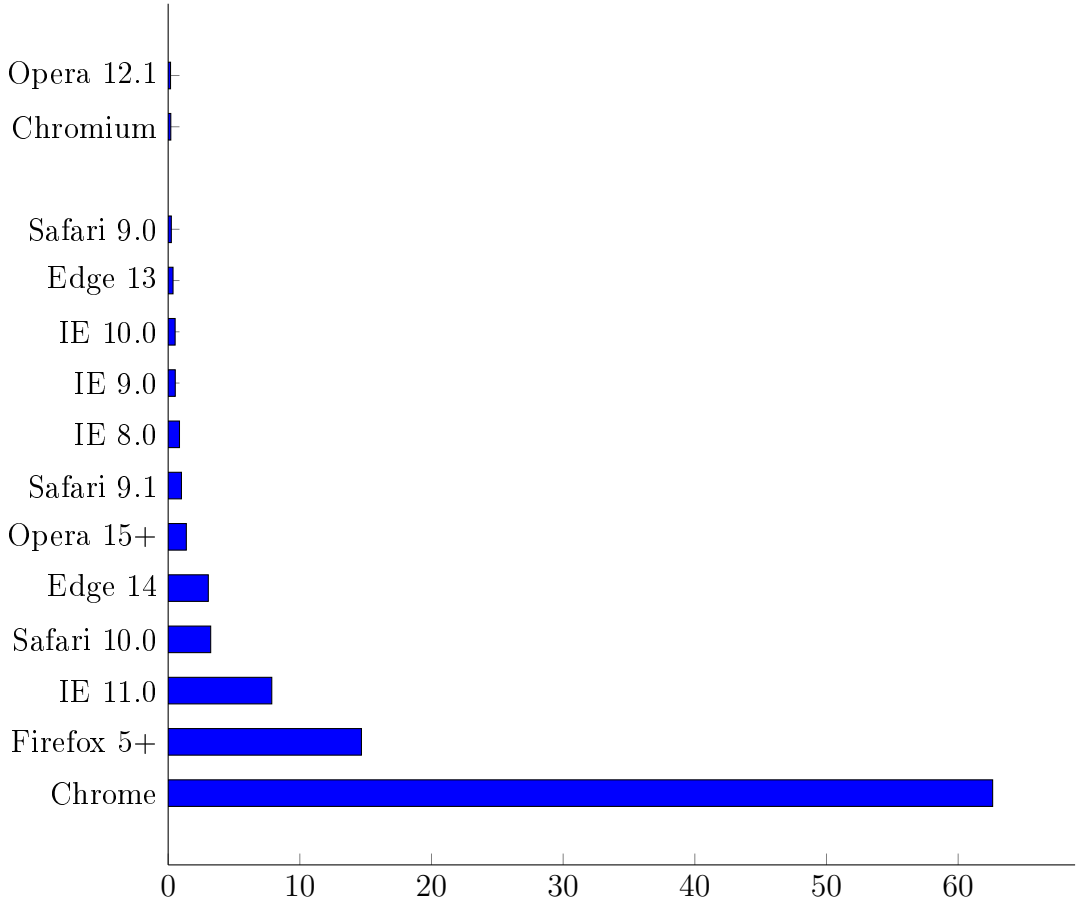
- **Vykreslování** – Klasické webové aplikace potřebují při většině uživatelských akcí překreslit celou stránku. To vede k pauze, kdy uživatel čeká na zpracování požadavku na serveru a poté vykreslení celé stránky webovým prohlížečem. Pokud je uživatel připojen pomalým připojením, nebo je server zaneprázdněný, může tato pauza vykreslování trvat řádově několik vteřin. V případě použití SPA to není potřeba. Pouze část stránky se vždy překreslí a často není potřeba k takové akci činnosti serveru.
- **Responzibilita** – Takovéto aplikace minimalizují reakční čas tím, že větší část logiky je přenesena k uživateli. Pouze zpracování dat, validaci a autentizaci má na starosti server. Jak jsou tato data zobrazena a přípaná filtrace je zpracována u uživatele pomocí SPA. Například pokud uživatel filtruje data pomocí výběru sloupce v grafu není potřeba provést získání dat, tím se nezatěžuje server a stránka zůstává nadále responzivní.
- **Notifikace** – Pokud takto napsaná aplikace musí čekat na server může uživateli dát nějakým grafickým prvkem najevo takovou zkutečnost. Případně je možné uživatele upozornit na zprávy ze serveru za pomocí **Webových notifikací**[4].
- **Přístupnost** – Díky tomu že jsou SPA napsány jako webové aplikace je možné se k nim dostat odkudkoliv, pokud má uživatel připojení k danému serveru, kde je takováto aplikace uložena.
- **Aktualizace** – V případě aplikací, které jsou uloženy u uživatele je často potřeba vydat aktualizační balíček a uživatel si ho musí nainstalovat. V případě

[3] Takovémuto přístupu se říká Service Worker a více se můžete dozvědět na `https://developers.google.com/web/fundamentals/getting-started/codelabs/offline/`

[4] Pro detailní popis můžete následovat `https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API/Using_the_Notifications_API`.

SPA tomu tak není, aktualizace se provádí takřka automaticky stačí na server nahrát novou verzi aplikace. Jediný problém je že si uživatel musí obnovit stránku.

- **Multiplatformní** – Webové aplikace jsou dostupné takřka na všech platformách, od mobilních zařízení až po velkoplošné obrazovky. Limitací je snad pouze používání zastaralých prohlížečů, naštěstí uživatelé začínají používat modernější prohlížeče a chápou nutnost aktualizací. Jak lze vidět na grafu 1.3, tak modernější prohlížeč chrome je o hodně používanější než zastaralý prohlížeč Internet explorer ve verzi 9 a méně. [19]



Obrázek 1.3: Podíl vybraných prohlížečů mezi uživateli. Vypracováno na základě dat TODO

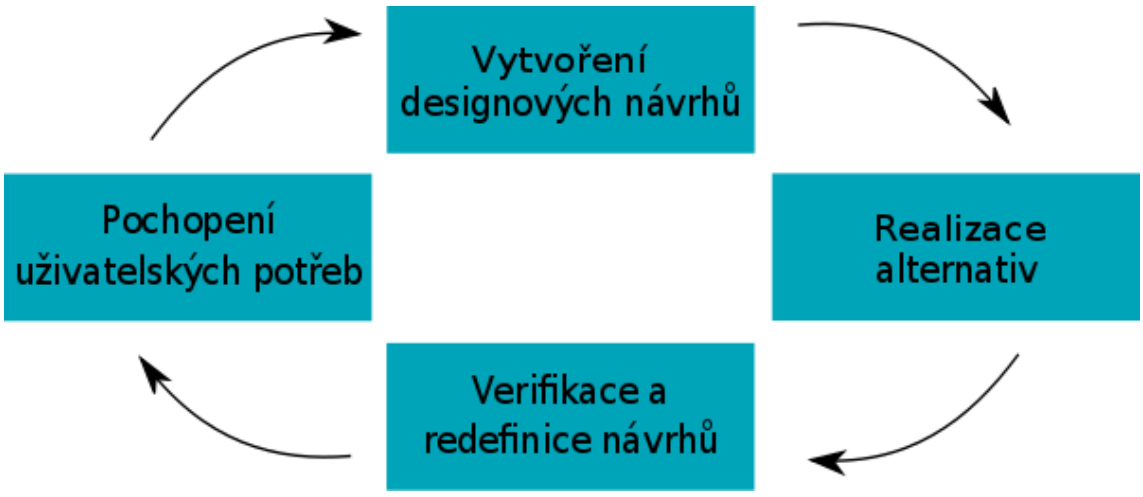
1.3.2 Dynamická a interaktivní vizualizace dat

Problém moderní doby je přehršel informací, proto se lidé snaží najít spůsob snadné vizualizace velkého množství dat. Jedním z možných způsobů jak vizualizovat data

je pomocí grafů (doslovně je to mapování informací na vizuální objekty), pokud použijeme webových technologií získáme silný nástroj v podobě zinteraktivnění takovýchto grafů. [20]

Statická vizualizace je často nepoužitelná a nedostatečná, proto je vhodné použít více grafů najednou, to může vést k nepřehlednosti a ke zmatení. Proto vznikla technologie interaktivní vizualizace, kdy uživatel samotný se snaží vybrat a zpřehlednit data k obrazu svému. interaktivní vizualizace se změnila pouze nepatrně od roku 1996, kdy Ben Shneiderman z Univerzity Marilandu pronesl „Nejdříve náhled, poté zoom a filtrování, na závěr detail na vyžádání."[20]

Při vytváření interaktivní vizualizaci dat se musíme zaměřit na 4 fáze návrhu a vývoje. **Vytvoření designových návrhů** – kdy navrhneme co a jak bude vypadat v našem systému. **Realizace alternativ** – zapracujeme návrhy a vytvoříme první prototypy. **Verifikace a redefinice návrhů** – kdy předložíme uživateli námi vytvořené řešení a zapíšeme si jeho reakce a chování. **Pochopení uživatelských potřeb** – ze zápisků z předešlého kroku a z požadavků uživatele vyvedeme závěry a opět se vracíme do prvního kroku vytvoření návrhů. Celý koloběh si můžeme prohlédnout na obrázku 1.4. [21]



Obrázek 1.4: Cyklus návrhu grafických prvků.

1.3.3 Webová služba RESTful

Od počátku webových služeb server používal takzvané sezení (anglicky *session*), uživateli se při každém načtení stránky přiřadil speciální token, který si server uložil

a pracoval s ním pro přístup k datům. Server takto často například ukládal historii navštívených stránek na dané webové stránce, případně filtrovaná data atd. Bohužel ukládání těchto dat vedlo k zahlcení paměti serveru a dlouhým odpovědím ze serveru (kdy server musel často poskládat velké množství dat dohromady). Na základě tohoto se zavedla bezestavová komunikace, kdy si server již nedrží stav o uživatelských akcích, ale odpovídá na všechny dotazy jako na nové sezení. Jedním kdo využívá tohoto bezestavového přístupu k uživatelům je architektický styl RESTful (zkráceně můžeme používat pouze REST). [22]

Autentizace

Největším problémem při přístupu k datům, pří použití REST je rozhodně způsob autentizace, server si nedrží přihlášené sezení, ale musí nějakým způsobem odlišit práva jednotlivých uživatelů. Na základě toho vzniklo několik druhů autentizace při použití REST. [22]

Základní přihlášení – Basic Auth Uživatel si nejdříve vytvoří učet (nebo je mu přidělen) a na základě toho zná své uživatelské jméno a heslo. Při každém dotazu na server uživatel pošle toto jméno a heslo a server vyhodnotí zda mu pošle data jaká chce. Tento přístup je jednoduchý, ale velice nebezpečný. Jméno a heslo se posílá jako text, který je pouze spojen pomocí **Base64** do jednoho textu. Pokud server používá nezabezpečené připojení HTTP útočník může toto heslo a jméno jednoduše získat. Při použití zabezpečeného připojení tento problém mizí. [22]

OAuth 1.0 Je zkratka Otevřené autentikace (Open authentication) ve verzi jedna. Při této metodě odpadá problém, kdy uživatel používá více zařízení k přístupu na danou stránku a jedno zařízení by bylo napadeno a uživatel by byl nucen změnit heslo pro všechny zařízení (kde by byl nucen znovu provést přihlášení). Tento problém je řešen tak, že každé zařízení má speciální přihlašovací token a pokud jedno zařízení začne vykazovat špatné chováníuživatel je schopen tomuto zařízení zrušit přístup. [22]

OAuth 2.0 Toto je vylepšený protokol a je momentálně nejvíce rozšířený. Ve zkratce to znamená to, že není nutné pro každé zařízení generovat specifický token, který je možno v případě napadení zařízení zrušit. Tento protokl využívá prostřed-

níka pro uchování klientských údajů. Je to v podstatě to, že služba požádá jiný server, který spravuje daného uživatele o přihlašovací token. Uživatel je přesměrován na adresu třetí serveru, kam zadá své údaje, server poté vyhodnotí správnost těchto údajů a službě která si vyžádala přihlašovací token odešle buď povolí nebo zakáže přihlášení. Výhodou je to, že pokud je uživatel jednou přihlášen na třetím serveru není nutné se znovu přihlašovat. A také pokud uživatel již nevěří službě která má přístup k přihlašovacímu tokenu, jednoduše jí zakáže přístup. [22]

1.4 Server pro řízení přístupu a identity

Mnoho firem stále váhá s přechodem z vlastního řešení (jak databázových jednotek, tak aplikací) do cloudu. Hlavní problém je často zabezpečení, firmy jednoduše nechtějí věřit jiné firmě s důvěrnými informacemi, že je ochrání před případným napadením. Naštěstí mnoho poskytovatelů cloudů si toto uvědomuje a tak nemalou část financí investují do robustního autentikačního formátu. [23]

Mechanismy používané pro zabezpečení dat v cloudu

- **RSA** – Pro přihlášení je potřeba HW token, který je synchronizován se servery cloudu.
- **Vícefaktorové** – Pokud se uživatel chce přihlásit dostane email nebo zprávu s pinem, který po jednom přihlášení, nebo po určitém čase, ztrácí platnost.
- **Single sign-on** – Uživatel je přihlášen ke druhé službě a ta jej autorizuje vůči aplikaci, kterou se chystá použít. Výhoda je ta, že není nutnost vždy zadávat přihlašovací údaje, při přístupu k aplikaci je vyžádán od druhé služby takzvaný JSON web token. [23]

1.4.1 JSON Web Token

Zkráceně JWT jsou standard pro autentizaci napříč aplikacemi. Takový token nezávisí na programovacím jazyce, nese veškeré informace (přihlašovací údaje a svůj podpis) a je jednoduše použitelný (může být součástí hlavičky nebo součástí dotazu). [24]

JWT je text, který je zahashován pomocí base64 a rozdělen do 3 částí:

Hlavička obsahuje typ a hashovací algoritmus použitý pro dekódování podpisu.

Náklad zde se nachází data, která služba poskytuje. Často jsou zde přihlašovací údaje (uživatelské jméno a čas vypršení), dále může obsahovat jakákoliv data chceme, například pohlaví uživatele, věk, email, atd.

Podpis je text, který vznikne spojením hlavičky a nákladu, který je poté zahashován pomocí algoritmu, který je určen v hlavičce tokenu. [25]

2 ANALYTICKÁ ČÁST

Před samotným vývojem platformy pro definici a zpracování dat musíme provést průzkum aktuálních technologií zaměřujících se na vývoj webových aplikací a jejich výběr. Výběr echnologií, které nám pomohou při vývoji je velice podstatný, protože nám urychlí dodání a co je dležitější, pozdější úprava bude značně jednodušší pokud zvolíme technologie, které nám takovýto pozdější vývoj usnadní.

Nejenom vývojovími technologiemi je potřeba se zabývat, ale také aktuálním stavem trhu a toho co uživatelé nejčasteji používají. Proto se v této části zaměříme také průzkumem trhu.

2.1 Webové technologie

Aktuální trend v používání webových prohlížečů můžeme vidět na grafu 1.3, kde jde jasně vidět že moderní prohlížeče v podání Chrome a Firefox převládají nad tolika vývojáří proklínaný a čím dál tím méně používaný Internet Explorer. dále si uživatelé již uvědomují, že aktualizace prohlížeče je pro zachování zabezpečení nutností a tak naštěstí webové prohlížeče pomalu začínají pořádně fungovat s novým standardem JavaScriptu nazvaný EcmaSript 6 (znáý též pod názvem ES2015 a ES6). [26]

Nový standard ES6 přináší mnoho vylepšení a mnoho optimalizací. Nicméně ani většina moderních prohlížečů nepodporuje 100 % tento standard, například prohlížeč **Google chrome** ve verzi 57 podporuje 97 % nového standardu, obdobně je na tom **Edge** (verze 15 podporuje 95 %) a **Firefox** (verze 52 zvládá 94%). [27]

2.1.1 Webový aplikační rámec

V aktuální době se mnoha vývojářům webových aplikací ověřili takzvané webové aplikační rámce. Dříve hojně využíváná knihovna **jQuery** má již mnoho nástupců jak v podobě knihoven, tak aplikačních rámců. Výhoda takových aplikačních rámců je že se stará v podstatě o veškerou těžkou a neustále se opakující práci a nechává programátorovi volnou ruku při realizaci samostatné aplikace. [28]

V předešlém odstavci bylo použito obou pojmů, jak JavaScriptové knihovny, tak aplikačního webového rámce. Tyto pojmy dost často vedou k hádkám a nedorozumění, kdy valná většina programátorů nerozumí rozdílům mezi aplikačním

rámcem a knihovnou.

- **JavaScriptová knihovna** slouží ke konání jednoho úkolu. Vezměme si například výrobu kávy, můžeme si postavit vodu na oheň ohřát ji, rozemlít zrnka kávy a tento prášek zalít horkou vodou. Každý jeden nástroj by představoval knihovnu.
- **Aplikační webový rámec** má na starosti veškerou práci a často zahrnuje přesně definovanou architekturu. Pokud si vezmeme příklad s kávou, aplikační webový rámec by byl kávovar, do kterého nalijeme vodu a nasypeme kávová zrna. [29]

Angular.js je plnohodnotný aplikační rámec, v aktuální době pravděpodobně nejpoužívanější. [1].

Strukturální webový aplikační rámec, pro dynamické webové aplikace. Jeho přední výhodou je že se soustřeďuje na tvoření jednostránkových aplikací. Takže často snižuje počet dat nutných pro načtení apracování s aplikací. První datum vydaní bylo v roce 2010 a v roce 2016 byla vydána verze 2, která umožňuje vytvářet uživatelské rozhraní jak pro webové, tak pro mobilní aplikace.[30]

React.js není plnohodnotný aplikační rámec, jako **Angular.js**, nicméně je velkým hráčem na poli vývoje webových aplikací. Jeho výhodou je jeho jednoduchost, nastavení aplikace a počáteční vývoj je velice jednoduchý, nicméně nedokáže tolik věcí co plnohodnotný rámec. Samostatný ovšem není příliš vhodný, potřebuje několik rozšíření, které z něho udělají silnějšího hráče, to ale vede k jeho yesložitění a častým problémům s nastavením. [31]

Vue.js je progresivní rámec, pro vytváření uživatelských rozhraní. Jeho tvůrci kladli důraz na jeho jednoduchost použití a inspirovali se v **React.js**, výhodou oproti zmíněnému má použití s webovou stránkou, kdy se nesnaží obcházet její vykreslování, ale pracuje přímo s html elementy. Dále je tento rámec také zaměřen na práci s jednostránkovou aplikací, takže je výborným kompromisem mezi **Angular.js** – který může být příliš náročný pro pochopení, a **React.js** – který může vést k

[1] Přesná čísla se určit nedají, nicméně více informací o popularitě se můžete dočist zde: `https://hackernoon.com/5-best-javascript-frameworks-in-2017-7a63b3870282#.ufk9gznfd`

zesložitění při použití s více rozšířeními. [32]

2.1.2 Silné vs. slabé typování

Pro vývoj responzivních webových aplikací se používá JavaScript, který je ale slabě typovaný. To znamená že proměnná nemá předem určený pevný typ a může ho během chodu aplikace měnit. To může mít za následek chyby spojené s očekávaným typem, který může být chybný. Takže například při očekávání čísla budeme sčítat, ale aplikace nám během jejího chodu vrátila text. Takový problém může vést až k pádu, případně zamrznutí aplikace. Proto vznikl způsob jak přivést silné typování do slabě typovaných jazyků, jejich zástupci jsou **Typescript** a **Flow**.

Typescript je podmnožinou JavaScriptu, to znamená že veškeré soubory jsou přeloženy do předem vybrané verze specifikace a ty jsou poté spuštěny v prohlížeči. Není tedy nutné nutit uživatele do používání jiného prohlížeče. Velkou výhodou je také to, že se tým okolo Typescriptu snaží co nejvíce sledovat trend vývoje JavaScriptu a tak přináší mnoho novinek ještě před tím, než je začnou používat prohlížeče. Takže je možné použít velkou část nových technologií, bez nutnosti psát ne zrovna příjemně čitelný kód. [33]

Flow je další způsob jak přivést silné typování do světa JavaScriptu, oproti **Typescriptu** má výhody, že problémy s implicitní deklarací, které mohou nastat během používání aplikace jsou lépe vyhodnoceny a programátor je o tomto informován již během překladu aplikace. Ale jeho nevýhodou je, že nesleduje takovou měrou nejnovější trendy a je často pozadu. Někdy schválně, protože pro překlad novějších definic do staršího použití existují další nástroje. [34]

2.1.3 Responzivní aplikační rámec

Pro usnadnění používání webových aplikací na jekémkoliv zařízení vzniká aktuálně mnoho knihoven a aplikačních rámců, které mají za úkol sjednotit design napříč několika aplikacemi a také jejich responzibilitu. To znamená že stejná aplikace se bude chovat a vypadat stejně bez ohledu na to, na jakém zařízení ji otevřeme (mobilní zařízení, tablet, počítač, televize, atd.)

Material design vznikl na popud zjednodušení a zpřehlednění uživatelského rozhraní, jeho hlavním zaměřením je dotek, hlas a kliknutí. Definuje tři pravidla **Material je metafora** – chytře využívat prostor a pohyb jednotlivých elementů. **Tučné, grafické, záměrné** – základními stavebními bloky jsou typografie, mřížka, místo, barva a použití obrázků. **Pohyb má význam** – pokud uživatel vykoná nějakou akci design mu napoví jaká akce se stane pohybem. [35]

Boostrap jeden z prvních responzivních rámců, který přivedl sjednocení uživatelského rozhraní napříč všemi platformami s největším důrazem na mobilní platformy. Dost často jsou vytvořeny webové aplikace, které nerespektují různé rozlyšení pro různé uživatele, tomuto se chtěl bootstrap vyhnout. Nyní nabízí velké množství doplňků a rozšíření, díky kterým z něj dělají rávem nejrozšířenější responzivní rámec. [36]

Patternfly se inspiroval bootsrapem a vytvořil vlastní responzivní rámec, dále nabízí několik widgetů pro zobrazování složitějších uživatelských dat. Jeho hlavním zaměřením je unifikovat jednotlivé aplikace ve firmě, tak aby měli stejný, ale unikátní design. Proto nabízí několik jednoduchých přístupů co a jak dělat pro dosažení co nejvíce podobného vzhledu napříč aplikacemi. [37]

2.2 Single sign-on

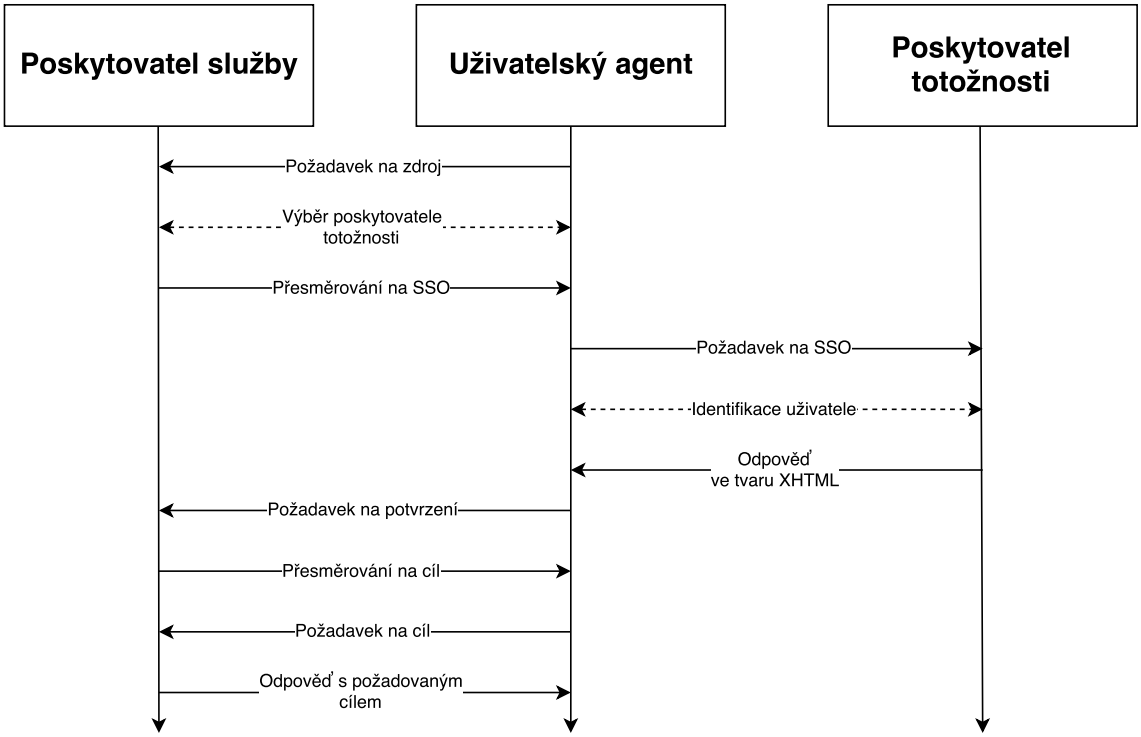
V aktuální době velká část společností nabízejících větší potrfolio aplikací volí takzvanou službu single sign-on, kdy uživatel nemusí pokaždé zadávat své přihlašovací údaje, ale o jeho identifikaci se postará specializovaná služba. Většinou se stačí do této služby přihlásit jednou a dokud nevyprší čas od posledního použití aplikace, která používá tuto službu, nebo není uživatel schválně odhlášen nemusí se znovu přihlašovat. Jako protiklad je postaven Single sign-off, kdy při odhlášení v jedné aplikaci je uživatel automaticky odhlášen ze všech ostatních aplikací, toto napomáhá dalšímu zabezpečení.

2.2.1 Standardy získání identity

V aktuální době jsou pravděpodobně nejrozšířenější dva standardy použití SSO – novější **OpenID Connect** a starší, ale robustnější **SAML**.

OpenID Connect je v podstatě identitní vrstva postavená nad OAuth 2.0, poskytuje možnost identifikace uživatele a také získání základních informací. Modernější a v aktuální době více používaný způsob převážně kvůli jeho jednoduchosti a nižšímu zatížení serveru. [38]

SAML se skládá ze dvou částí, poskytovatele služby a poskytovatele totožnosti (tím může být například Facebook, Google, Github, atd.). Jakým způsobem je uživatel přihlášen do služby můžeme vidět na obrázku 2.1. Nejdříve uživatel vytvoří požadavek na poskytovatele služby, ten vygeneruje SAML požadavek v URL, aplikace přesměruje uživatele na poskytovatele totožnosti, ten ověří přihlašovací údaje, přepošle zpět na poskytovatele služby požadavek na potvrzení, dále je uživatel přesměrován na cílový požadavake, o který si znovu požádá a server mu jej vrátí. [39]



Obrázek 2.1: Předání uživatelského požadavku při použití SAML2

2.2.2 Autentikační servery

Aktuální trh nabízí velké množství autentikačních serverů, které spravují uživatele a poskytují přihlášení do více aplikací. Některé jsou dokonce volně k použití bez

nutnosti zakoupit licenci, je nutné pouze spravovat servery na kterých takováto služba poběží.

Keycloak je open sourcové řešení, které nabízí připojení jak přes sociální síťe a možnost získání uživatelských účtů z LDAP, Active Directory a nebo přímo z databáze. Jeho velkou premisou je jednoduchost správy uživatelů, včetně definice přístupových práv pro jednotlivé části aplikace. [40]

Sterling externí autentikační server je řešení od IBM nabízející rozšířené autentizace a validace služeb pro IBM produkty, nabízí napojení například na LDAP a možnost SSH autentizace. [41]

Active Directory Federation Services od firmy Microsoft nabízí možnost sdílet identifikační informace napříč důveryhodným obchodním partnerům skrze extranet. Výhodou je že se nemusí aplikace starat o všechny uživatele, ale pouze si vyžádá od spřátelené aplikace jeho práva. [42]

2.3 Datová analýza

Pokud firma disponuje velkým množstvím dat je vhodné nad těmito daty najít nějaké souvislosti, které jí dopomůžou k vytváření dalšího zisku, případně pouze pro pochopení daného zákazníka, nebo skupiny zákazníků. Pokud je těchto dat opravdu hodně, můžeme je označit za Big Data (popsáno v 1.2.1), poté procesům k získání dalších informací říkáme Dolování dat (blíže popsáno v 1.2.2).

2.3.1 Shluková analýza

Pro bližší pochopení velkého množství dat je vhodné použít metodu shlukoví analýzy, což je zjednodušené proces organizace objektů do skupin, tak aby její členové byli nějakým způsobem co nejvíce podobní. Shlukovou analýzu můžeme rozdělit do dvou skupin **nehierarchické** a **hierarchické**.

Nehierarchické metody mohou být tvrdé nebo jemné, tvrdé seskupují data na základě specifické shlukové vlastnosti – pokud patří do tohoto shluku namůže patřit do jiného. Jemné na druhou stranu nedefinují přesnou hranici – patří do tohoto shluku, ale má některé vlatnosti daotových bodů jiných shluků. Pro dělení můžeme použít metodu K-means, která přiřadí každý bod do shluku jemuž středu je nejblíže a

při každém běhu algoritmu se středy shluků přepočítávají jako aritmetické průměry všech bodů.

Hierarchické metody můžeme použít ve dvou způsobech ze shora dolů a odspodu nahoru. V prnvím případě vezmeme všechny datové body a označíme jej za shluk, který poté rozdělíme na další shluky, které dále dělíme. Opět můžeme využít K-means pro dělení shluků. Druhý způsob (odspodu nahoru) je nejlépe použít v případě že máme menší vzorek dat a chceme co nejlepší shluk.

Pro shrnutí je potřeba si uvědomit dvě věci, jaký je nejmenší počet shluků a naopak jaký je největzší počet těchto shluků. Na první otázku je jednoduché odpovědět, je to v podstatě celý seznam datových bodů v jednom shluku (není nám to příliš užitečné, ale je to shluk dat). Na otázku největšího počtu shluků můžeme říct, že jím je počet všech datových bodů (toto opět není příliš užitečné). Proces shlukování je tedy rozdělování těchto dat do shluků větších jak jedna a menších než počet prvků, proces ukončíme v momenté, kdy jsme spokojeni s celkovým počtem shluků, případně velikostí těchto skupin.

2.3.2 Rozhodovací stromy

Tento algoritmus slouží k získání pravidel a vztahů v datovém souboru pomocí větvení. Díky použití stromů je tento algoritmus rychlý a výhodný pro použití s počítačem. Složení stromu je takové že na vrcholu je jeden uzel, kterému se říká kořen, ze kterého vede několik hran spojující vnitřní uzly, které mají opět několik hran projujících další uzly. Každý uzel musí mít právě jednoho předka (ne více ani méně), kromě kořenového uzlu, který nemá žádného předka. Výhodou je jednoduchost, efektivnost a možnost použít i pro velký objem dat, na druhou stranu pokud nám budou některá data chybět, nebo budeme mít spojitá data rozhodovací stromy budou mít problémy. Při výběru vlastnosti, která je nejvíce odlišná od ostatních příkladů v ostatních třídách se požívá takzvané entropie, její výpočet lze vidět na 2.1, kde p_t je pravděpodobnost výskytu třídy t a t je počet tříd

$$E(S) = -\sum_{t=1}^{T}(p_t log_2 p_t) \tag{2.1}$$

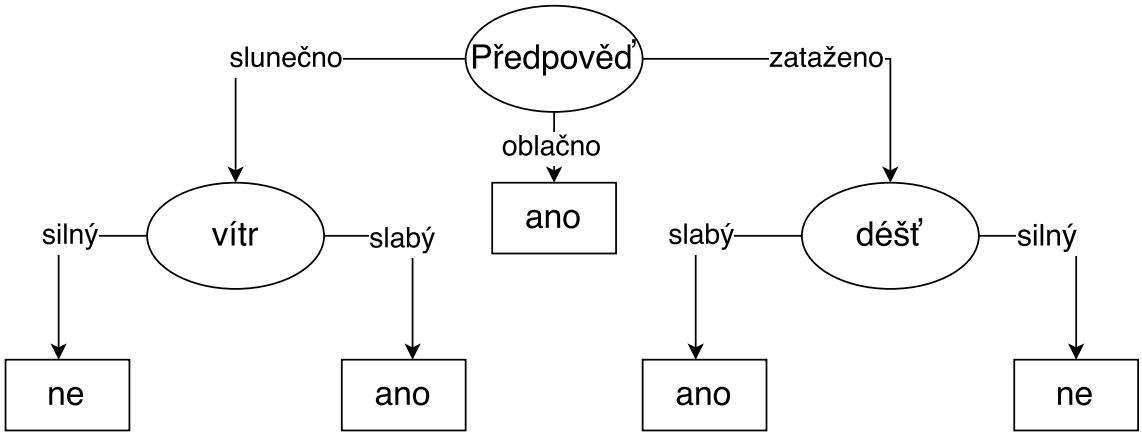
Pro vytvoření stromu se vezme tabulka hodnot a pro každý atribut se vypočítá jeho entropie, ta znázorňuje homogenitu prvku. Pokud je prvek naprosto homo-

genní (nácházející se ve všech atributech) jeho entropie je 0, naopak pokud je prvek naprosto heterogenní (nachází se pouze v jednom atributu) jeho entropie je 1. Při prvnotním vytvoření rozhodovacího stromu je tedy potřeba mít co nejlepší trénovací data, nicméně velké množství dat může mít za následek příliš složitý strom, který je opět nepoužitelný (proto je potřeba dbát na vhodnou velikost učebních dat). Pokud nám vznikne složitý a ne příliš efektivní rozhodovací strom je vhodné použít takzvané prořezávání, kdy odstraníme nadbytečné podstromy (velice vhodné v případě že jsme použili velký vzorek dat pro vytvoření stromu).

Příklad algoritmu, který se používá pro vytváření rozhodovacích stromů je takzvaný ID3, je založen na booleanovských hodnotách, používá hladové vyhledávání v prostoru [2] a strom je vytvářen odshora dolů. Vstupní parametry tohoto algoritmu jsou **příklady** – data na kterých je rozhodovací strom vytrénován **atributy** – nad nimiž ude strom testován a **cílový atribut** – tento atribut se bude vytvořený strom snažit predikovat. ID3 algoritmus byl několikrát vylepšen a upraven pro použití za různých podmínek jako například C4.5 (vhodné při chybejících, nebo spojitých datech), C5 (vylepšený C4.5), pro vytvoření binárních rozhodovacích stromů je například vhodné použít klasifikační a regresní stromy (CART), případně pro práci s opravdu velkým množstvím dat je možné použít algoritmus SPRINT.

Na obrázku 2.2 můžeme vidět příklad stromu, který vznikl pro rozhodování zda jít hrát badminton, pro jeho vytvoření byl použit algoritmus ID3 s menším vzorekem dat.

[2] Pro bližší informace o hladovém prohledávání `http://www.how2examples.com/artificial-intelligence/tree-search`



Obrázek 2.2: Příklad rozhodovacího stromu při použití algoritmu ID3.

3 VLASTNÍ NÁVRHY

text[?][?]

ZÁVĚR

Seznam obrázků

Seznam tabulek

Literatura

[1] MARVIN, Rob. Building an App With No Coding: Myth or Reality. *Pcmag*[online]. Ziff Davis, LLC. PCMag Digital Group, 2016 [cit. 2017-03-11]. Dostupné z: http://www.pcmag.com/article/345661/building-an-app-with-no-coding-myth-or-reality

[2] RUBENS, Paul. Use Low-Code Platforms to Develop the Apps Customers Want. *CIO* [online]. IDG Communications, 2014 [cit. 2017-03-11]. Dostupné z: http://www.cio.com/article/2845378/development-tools/use-low-code-platforms-to-develop-the-apps-customers-want.html

[3] MARVIN, Rob. How low-code development seeks to accelerate software delivery. *SD Times* [online]. BZ Media LLC., 2014 [cit. 2017-03-11]. Dostupné z: http://sdtimes.com/low-code-development-seeks-accelerate-software-delivery/

[4] Zoho Creator REVIEW. *Finances Online* [online]. [cit. 2017-03-11]. Dostupné z: https://reviews.financesonline.com/p/zoho-creator/

[5] CIOT, Thierry. What is a Low-Code Platform? *Progress* [online]. 2016 [cit. 2017-03-11]. Dostupné z: https://www.progress.com/blogs/what-is-a-low-code-platform

[6] OpenShift Origin Overview. *OpenShift Origin* [online]. [cit. 2017-03-11]. Dostupné z: https://docs.openshift.org/latest/architecture/index.html

[7] HURWITZ, Judith. *Cloud computing for dummies*. Hoboken, NJ: Wiley Pub., c2010. ISBN 978-0470484708.

[8] ROME, C. H. *The cloud computing Book: The ultimate guide to mastering cloud computing*. Fifth edition. Bernemouth: Imagine Publishing, 2015.

[9] CHANDRASEKARAN, K. *Essentials of Cloud Computing*. Maiami: CRC Press, 2014. ISBN 978-1482205435.

[10] Co je PaaS?: Platforma jako služba. *Microsoft Azure* [online]. [cit. 2017-03-12]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-paas/

[11] TODO data science business

[12] TODO predctive analytics

[13] TODO big data analytics

[14] TODO big data for dummies

[15] BRAMER, Max. *Principles of data mining*. 3rd edition. 2016. ISBN 978-1-4471-7306-9.

[16] TODO data mining principles

[17] TODO the forrester wave advanced data visualization adv platforms q3 2012

[18] TODO Serverless Single Page Apps

[19] TODO 1617290750Single

[20] TODO interactive-data-reily

[21] HARTSON, H. Rex. a Pardha S. PYLA. *The UX Book: process and guidelines for ensuring a quality user experience*. Boston: Elsevier, c2012. ISBN 978-0123852410.

[22] ALLAMARAJU, Subrahmanyam. *RESTful Web services cookbook*. Sebastopol,CA.: O'Reilly, c2010. ISBN 978-0596801687.

[23] TODO cloud security

[24] TODO `https://scotch.io/tutorials/the-anatomy-of-a-json-web-token`

[25] TODO RFC `https://tools.ietf.org/html/rfc7519`

[26] TODO `https://benmccormick.org/2015/09/14/es5-es6-es2016-es-next-whats-going-on-with-javascript-versioning/`

[27] TODO `https://kangax.github.io/compat-table/es6/`

[28] TODO `https://medium.com/javascript-scene/top-javascript-frameworks-topics-to-learn-in-2017-700a397b711#.fng4s5tee`

[29] TODO `http://voidcanvas.com/framework-vs-library-one-better/`

[30] TODO `https://docs.angularjs.org/guide/introduction`

[31] TODO `http://www.developer.com/services/what-is-react.js.html`

[32] TODO `https://vuejs.org/v2/guide/`

[33] TODO `http://haroldrv.com/2015/02/typescript-making-javascript-strongly-type`

[34] TODO `https://code.facebook.com/posts/1505962329687926/`

flow-a-new-static-type-checker-for-javascript/

[35] TODO https://material.io/guidelines/#introduction-goals

[36] TODO https://www.taniarascia.com/what-is-bootstrap-and-how-do-i-use-it/

[37] TODO https://www.patternfly.org/get-started/frequently-asked-questions/

[38] TODO http://openid.net/connect/

[39] TODO http://www.gigya.com/blog/the-basics-of-saml/

[40] TODO http://www.keycloak.org/about.html

[41] TODO https://www.ibm.com/support/knowledgecenter/en/SS4T7T_2.4.1/com.ibm.help.seasimplementationguide.doc/SEAS_About_SEAS.html

[42] TODO https://msdn.microsoft.com/en-us/library/bb897402.aspx

[43] MARZ, Nathan a James WARREN. *Big data: principles and best practices of scalable real-time data systems*. ISBN 978-1617290343.

[44] HAN, Jiawei, Micheline KAMBER a Jian PEI *Data mining: concepts and techniques*. 3rd ed. Haryana, India ; Burlington, MA: Elsevier, 2012. ISBN 9789380931913.

[45] SADALAGE, Pramod J. a Martin FOWLER *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Upper Saddle River, NJ: Addison-Wesley, c2013. ISBN 978-0321826626.