UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2019 Bc. Radim Bednář

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Webová aplikace pro hudební fanoušky

Bc. Radim Bednář

Diplomová práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2018/2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Radim Bednář**
Osobní číslo:	**I17202**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Webová aplikace pro hudební fanoušky**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem práce bude vytvořit webovou aplikaci pro hudební fanoušky. Aplikace by měla fungovat mimojiné i jako doporučovací systém, který dokáže hudebnímu fanouškovi podle jeho preferencí doporučovat nadcházející kulturní akce. Aplikace bude implementována na platformě Java ve frameworku Spring (backend) a Javascript (frontend) s přihlédnutím na uživatelskou přívětivost. Text práce bude kromě samotné problematiky obsahovat i přehled použitých technologií, rešerši o existujících alternativách a analýzu s realizací aplikace. Dále by neměl chybět nástin logiky doporučovacího mechanismu.

Rozsah grafických prací:

Rozsah pracovní zprávy: **50-60 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

WALLS, Craig. Spring in action. Fourth Edition. Shelter Island, NY: Manning, [2015]. ISBN 978-1617291203 PETR, Pavel. Data Mining. Vyd. 3. Pardubice: Univerzita Pardubice, 2010-. ISBN 978-80-7395-325-6 JANNACH, Dietmar. Recommender systems: an introduction. New York: Cambridge University Press, 2011. ISBN 978-0521493369 AGGARWAL, Charu C. Recommender systems: The Textbook. New York, NY: Springer Science+Business Media, 2016. ISBN 978-3319296579 MARTIN, Robert C. Čistý kód: [návrhové vzory, refaktorování, testování a další techniky agilního programování]. Brno: Computer Press, 2009. ISBN 978-80-251-2285-3 KRUG, Steve. Don't make me think!: a common sense approach to Web usability. 2nd ed. Berkeley, Calif: New Riders Pub., c2006. ISBN 978-0321344755

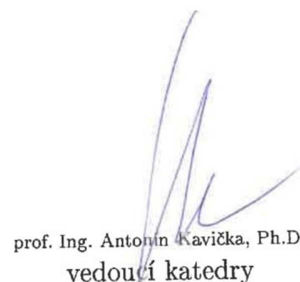
Vedoucí diplomové práce: **Ing. Jan Merta**
Katedra řízení procesů

Datum zadání diplomové práce: **22. října 2018**

Termín odevzdání diplomové práce: **18. května 2019**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.



prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 17. listopadu 2018

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 19. 8. 2019

Radim Bednář

PODĚKOVÁNÍ

Na tomto místě bych chtěl poděkovat svému vedoucímu Ing. Janu Mertovi za důkladné vedení mé diplomové práce, dále také rodině a přátelům za podporu při studiu, a především členům NinjaAreny za nekonečnou podporu a motivaci během magisterského studia.

ANOTACE

Diplomová práce se zabývá problematikou doporučovacích systémů. V rámci toho je popsána funkce doporučovacího systému včetně možností doporučování. Výsledkem práce je aplikace rozdělená do dvou částí. První je REST API napsané v Java frameworku SpringBoot jako backend a druhou částí je aplikace napsaná v Javascript frameworku React.js jako frontend. Aplikace je pro hudební fanoušky a jedná se o doporučovací systém pro události v okolí.

KLÍČOVÁ SLOVA

Webová aplikace, doporučovací systém, Java, SpringBoot, Spring, React.js, Redux

TITLE

Web Application for Music Fans

ANNOTATION

The diploma thesis deals with the issue of recommender systems. It describes the recommender system function including referral options. It results in an application divided into two parts. The first part is the REST API written in the SpringBoot Java framework as the backend and the second part is an application written in the React.js JavaScript framework as the frontend. The application target group are the music fans and it is a recommender system for music events nearby the user.

KEYWORDS

Web application, recommender system, Java, SpringBoot, Spring, React.js, Redux

OBSAH

Úvod 15

1 Aplikace pro hudební fanoušky 16

1.1 Rozdělení aplikací 16

1.1.1 Aplikace pro skládání hudby 16

1.1.2 Aplikace pro zpracování hudby 16

1.1.3 Aplikace pro konečné uživatele 16

1.2 Aplikace pro vyhledávání koncertů 17

1.2.1 Existující alternativy 17

1.2.2 Ostatní aplikace 18

2 Doporučovací systémy 19

2.1 Skupiny doporučovacích systémů 19

2.1.1 Kolektivní filtrování (Collaborative recommendation) 19

2.1.2 Filtrování na základě obsahu (Content-based recommendation) 20

2.1.3 Filtrování na základě zkušeností (Knowledge-based recommendation) 20

2.1.4 Hybridní systémy 21

2.2 Doporučování na základě obsahu 21

2.2.1 Reprezentace obsahu a podobnost obsahu 21

2.2.2 Vyhledávání na základě podobnosti 22

2.3 Shrnutí 24

3 Použité technologie 25

3.1 Spring framework 25

3.1.1 Dependency injection 26

3.1.2 Aspektově-orientované programování 26

3.1.3 Beans 26

3.1.4 SpringBoot 27

3.1.5 Manažer pro build aplikace 28

3.2 REST 29

3.2.1 Klient-server 30

3.2.2 Bezstavovost 30

3.2.3 Využití mezipaměti 30

3.2.4 Jednotné rozhraní 31
3.2.5 Vrstvený systém 31
3.2.6 Kód na vyžádání 31
3.2.7 XML vs JSON 31
3.3 React 32
3.3.1 Virtual DOM 32
3.3.2 JSX 32
3.3.3 React komponenty 33
3.3.4 Životní cyklus komponenty 34
3.3.5 ES6 35
3.3.6 Funkce 36
3.3.7 Seznamy 36
3.3.8 Události 36
4 Analýza webové aplikace 38
4.1 MVC 38
4.2 Funkční a nefunkční požadavky 39
4.2.1 Funkční požadavky 39
4.2.2 Nefunkční požadavky 40
4.3 Případy užití 40
4.4 UML 41
4.5 Model analytických tříd 42
4.6 Datový model 43
5 Implementace webové aplikace 45
5.1 Backend 45
5.1.1 RESTful API 45
5.1.2 SpringBoot Security 46
5.1.3 SpringBoot JPA 48
5.1.4 MySQL 51
5.1.5 Doporučování událostí 52
5.1.6 Plánovač 55
5.2 Frontend 56

5.2.1 Routes ... 58
5.2.2 Redux ... 58
5.2.3 Axios interceptor ... 60
5.2.4 Přihlašování a registrace ... 61
5.2.5 Profil uživatele ... 61
5.2.6 Vytvoření události ... 62
5.2.7 Registrace k události ... 62
5.2.8 Vložení interpreta ... 63
5.2.9 Administrace ... 63

Závěr ... 64

Použitá literatura ... 65

Přílohy ... 67

SEZNAM OBRÁZKŮ

Obrázek 3.1: Rozdělení Spring framework modulů. Zdroj: převzato z [4] 25
Obrázek 3.2: Ukázka stromové struktury. Zdroj: autor 33
Obrázek 3.3: Graf životního cyklu komponenty. Zdroj: autor..35
Obrázek 4.1: Graf MVC modelu. Zdroj: autor 39
Obrázek 4.2: Diagram případů užití. Zdroj: autor..41
Obrázek 4.3: Ukázka UML – editace profilu. Zdroj: autor..42
Obrázek 4.4: Model analytických tříd. Zdroj: autor 42
Obrázek 4.5: Datový model aplikace. Zdroj: autor...44
Obrázek 5.1: Ukázka funkcionality axios interceptoru. Zdroj: autor.................................60
Obrázek 5.2: Registrační a přihlašovací formulář. Zdroj: autor 61
Obrázek 5.3: Ukázka karty doporučené události. Zdroj: autor 62

SEZNAM TABULEK

Tabulka 2.1: Seznam knih v databázi 22
Tabulka 2.2: Preference uživatele 22
Tabulka 3.1: Možnost zápisu typu těla HTTP zprávy...31
Tabulka 5.1: Přehled základních anotací pro entity 49
Tabulka 5.2: Filtry pro organizátora 52
Tabulka 5.3: Filtry pro fanouška...53

SEZNAM UKÁZEK KÓDU

Ukázka kódu 3.1: Ukázka nastavení pro databázi v souboru application.properties..........28
Ukázka kódu 3.2: Ukázka importu knihovny pomocí Maven 29
Ukázka kódu 3.3: Ukázka funkce JSX...33
Ukázka kódu 3.4: Ukázka změny stavu komponenty 33
Ukázka kódu 3.5: Ukázka vytvoření a použití komponenty..34
Ukázka kódu 3.6: Ukázka zápisu funkce 36
Ukázka kódu 3.7: Ukázka použití funkce map 36
Ukázka kódu 3.8: Definice a použití zachycení události po stisku tlačítka........................37
Ukázka kódu 5.1: Přetížení metody configure..46
Ukázka kódu 5.2: Přetížení metody loadUserByUsername...47
Ukázka kódu 5.3: Definice třídy Genre 49
Ukázka kódu 5.4: Porovnání metod pro dotazování do databáze 51
Ukázka kódu 5.5: Definice funkce distance pro vypočítání vzdálenost mezi dvěma body.52
Ukázka kódu 5.6: Zdrojový kód metody pro doporučení událostí dle žánru......................54
Ukázka kódu 5.7: Ukázka výpočtu priority 55
Ukázka kódu 5.8: Nastavení plánování...56
Ukázka kódu 5.9: Ukázka použití služby axios k volání REST API 57
Ukázka kódu 5.10: Ukázka vytvoření úvodní stromové struktury react aplikace 57
Ukázka kódu 5.11: Ukázka obsahu komponenty App 58
Ukázka kódu 5.12: Zobrazení připojení příznaku isAuth do props ze store včetně použití 59
Ukázka kódu 5.13: Ukázka definice akčních typů...59
Ukázka kódu 5.14: Ukázka definice metody, vracející akční typ reduktoru a její vyvolání59
Ukázka kódu 5.15: Ukázka definice reduktoru..60

SEZNAM ZKRATEK

AOP	Aspekt-Oriented Programming
API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
CSRF	Cross-site Request Forgery
CSS	Cascade Style Sheet
DI	Dependency Injection
DOM	Document Object Model
HTTP	Hypertext Transfer Protocol
JPA	Java Persistant Api
JS	JavaScript
JSON	JavaScript Object Notation
JWT	Java Web Token
MVC	Model View Controller
ORM	Object-Relation Mapping
PDF	Portable Document Format
REST	REpresentional State Transfer
SB	Spring Boot
TS	TypeScript
UML	Unified Model Language

Typografické konvence

V textu se budou vyskytovat slova či slovní spojení která jsou zvýrazněny kurzívou. Jedná se o odborné názvy v teoretické části nebo názvy proměnných, metod či souborů v části, kde je popisována aplikace.

Obrázky, tabulky a ukázky kódů jsou očíslovány podle jejich výskytu v dané kapitole.

ÚVOD

V dnešní moderní době nás hudba i svět internetu obklopuje ze všech stran. Webové aplikace ve většině případů nahrazují dříve oblíbené aplikace desktopové z důvodu uživatelské přístupnosti a nabízejí různá uživatelská rozhraní. Standardem se tak v dnešní době stávají responsivní, dynamické aplikace zaměřené na uživatelskou přístupnost a vizualitu. Dopředu jde ale také hudba. Existuje čím dál více kapel různých složení a žánrů a z toho plynoucí i nabídka, ale i poptávka po koncertech.

Cílem této práce bude tedy nastínění problematiky vývoje webových aplikací a zajištění nabídky i poptávky po koncertech různých žánrů pro hudební fanoušky podle jejich preferencí, ale zároveň potencionální pořadatele dle výskytu fanoušků konkrétního hudebního žánru či interpreta v okolí. Aplikace tedy bude pracovat s uživatelskými preferencemi, a především GPS souřadnicemi, pomocí kterých vyhodnotí nabídku existujícího koncertu, či nabídky pořádání koncertu v okolí.

Pro vývoj webových aplikací se používá různých frameworků jak backendových, tak frontendových. Neexistuje univerzální jazyk, pomocí kterého by šla celá aplikace naprogramovat. Je potřeba využít více programovacích jazyků, které musí být navzájem kompatibilní, aby bylo možné vytvořit robustní, výkonnou a uživatelsky přívětivou aplikaci. V rámci této právě bude přiblížen Java framework SpringBoot, který je využit na backend a JS knihovna React.js, pomocí které je naprogramovaný frontend. Zároveň bude pro komunikace mezi backendem a frontendem využito webové služby REST, která je postavena na HTTP požadavcích a v neposlední řadě využití Google API pro práci se souřadnicovými daty.

V úvodních dvou kapitolách bude představena problematika aplikace pro hudební fanoušky, včetně již existujících alternativ. Třetí kapitola bude věnována podrobnému přiblížení doporučovacího systému. Čtvrtá kapitola bude popisovat technologie použité pro vývoj aplikací. Poslední dvě kapitoly budou věnované analýze a návrhu aplikace a jejímu programovému zpracování včetně nastínění logiky doporučovacího algoritmu.

1 APLIKACE PRO HUDEBNÍ FANOUŠKY

Aplikace pro hudební fanoušky je aplikace, která uživatele jakýmkoliv způsobem spojuje s hudbou. Může se jednat o aplikaci, která hudbu produkuje, ale také takovou, která ji zpracovává nebo uživateli hudbu přibližuje. Může se jednat jak o malou aplikaci pro konverzi hudebních formátů, tak o robustní aplikaci, kdy se může jednat o celé hudební studio.

1.1 Rozdělení aplikací

Aplikace lze rozdělit do několika skupin podle toho, jakým způsobem s hudbou pracují.

1.1.1 Aplikace pro skládání hudby

Aplikace produkující hudbu jsou takové aplikace, které uživateli umožní pomocí vlastních nástrojů vytvořit hudební nahrávku. Jedná se o programy, které umožňují skládání různých hudebních či zvukových nahrávek za sebou či jejich mixování a následné editování. Výsledkem takovéto aplikace pak může být tvorba vlastní hudby například jako doprovodná hudba k videu, znělky apod. Tato skupina zahrnuje jednoduché nástroje pro uživatele, kteří se se zpracováním hudby teprve seznamují, ale i programy pro profesionální uživatele, kteří se zpracováním hudby již živí.

Jako příklad lze uvést třeba MAGIX Music Maker

1.1.2 Aplikace pro zpracování hudby

Do této kategorie spadají programy a aplikace, které dokáží zpracovat celé hudební nahrávky pro lepší zážitek z poslechu. Dokáží pracovat s frekvencemi, několika hudebními kanály a mnoho dalšího. Jedná se o aplikace, které používají především profesionální nahrávací studia pro zpracování nahrávek k vytvoření kvalitní hudby například pro vydání nové hudební desky. Tyto aplikace již vyžadují zkušenosti práce s hudbou, neboť dokáží odstraňovat nežádoucí prvky jako jsou šum, zvuky v pozadí apod. a to již vyžaduje mnohé znalosti o vlastnostech zvuku, zvukových frekvencích, signálech a mnohé techniky, jak těchto úprav docílit.

Příkladem může být profesionální nástroj Adobe Audition.

1.1.3 Aplikace pro konečné uživatele

Do této kategorie lze zařadit aplikace, které využívají již koneční uživatelé, tedy již skutečně hudební fanoušci, kteří hudbu nezpracovávají. Jedná se o různé přehrávače hudby (např. Winamp), aplikace na rozpoznání hudby (např. Shazam), ale také různé hudební portály věnované hudbě a její produkci. Jedná se například o webovou aplikaci Bandzone, což je nejrozsáhlejší komunita kapel a jejich fanoušků na českém internetu či aplikace, které shromažďují informace o pořádaných koncertech, a právě tyto aplikace budou podrobněji

zpracované v následující kapitole. Další velkou aplikací, která lidem přibližuje hudbu je video portál YouTubes klipy k jednotlivým skladbám a v neposlední řadě také hudební aplikace Spotify, což je jeden z předních světových hudebních portálů na sdílení a poslech hudby, disponující sofistikovaným doporučovacím systémem, který uživatelům doporučí hudbu přesně na míru v závislosti na tom, co rádi poslouchají, jakou mají náladu, při jaké příležitosti se nachází apod.

1.2 Aplikace pro vyhledávání koncertů

Aplikace pro vyhledávání koncertů je aplikace, která slouží uživatelům pro vyhledání konkrétního koncertu pomocí jím zadaných kritérií. V případě, že uživatel navštíví nějakou aplikaci, která toto vyhledávání nabízí, tak nejčastěji uvidí nějaké vyhledané koncerty podle základního filtru nastaveného aplikací ve výchozím stavu. Tento filtr ovšem aplikace umožňuje změnit, což uživateli dává možnost nechat si vyhledat koncert na míru.

1.2.1 Existující alternativy

Na internetu existuje celá řada webových aplikací, ve kterých uživatel nalezne informace o koncertu konaném v budoucnosti. Tyto aplikace pracují na různém principu vyhledávání a uživateli tak nabízí různé výsledky. Může se jednat o aplikace, které jsou zaměřeny na velké koncerty a hudební festivaly, které například lákají na turné hvězd zahraniční či české hudební scény. Dále to mohou být aplikace, které naopak podporují právě začínající interprety, aby se dostali do podvědomí lidí a měli tak větší šanci se prosadit. Takové aplikace pak mohou využívat například hudební kluby ve městech. Dále to mohou být aplikace, které pracují na nějakém složitějším algoritmu, který zpracovává uživatelský profil a nabízí mu tak více personalizované výsledky. Ve většině případů pak tyto aplikace odkazují na aplikace typu „ticketportal“, kde si uživatel může rovnou koupit lístek, či disponují vlastním systémem pro prodej a výdej lístků.

Goout

Jednou z aplikací, která nabízí uživatelem různé události je aplikace goout.net. Tato aplikace neslouží pouze k vyhledávání koncertů, ale kulturních akcí obecně, jako jsou například různé výstavy, kina, divadla apod. Aplikace nedisponuje „doporučovacím systémem“, ale nabízí možnosti jak pro uživatele, tak pro organizátory. Po registraci si zde lze zařadit oblíbené umělce, či různé události a dále je sledovat kvůli různým novinkám apod. Aplikace poté zobrazuje například různé koncerty daných umělců, mezi kterými lze použít různé filtry na

detailnější výběr. Aplikace nabízí i zakoupení vstupenek. Pro organizátory nabízí možnosti vytvořit různorodé události a dostat je tak do povědomí lidí v okolí měst Praha, Brno, Plzeň, Ostrava, Bratislava, Berlín, Varšava, Krakov, Poznaň a Vratislav. Aplikace je dostupná na odkazu https://goout.net.

SongKick

Další z aplikací nabízející události v okolí je aplikace SongKick, která již disponuje doporučovacím systémem. Aplikace umožňuje si v profilu uložit seznam lokací, pro které bude aplikace zobrazovat koncerty interpretů, které uživatel sleduje. Tato aplikace disponuje kolaborativním doporučováním, které je využito pro doporučování dalších interpretů, které si oblíbili fanoušci sledující stejné interprety. Aplikace je celosvětová, a tedy nabízí sledování lokací po celém světě. Pokud uživatel má přidané některé interprety, kteří v dohledné době nemají naplánovaný koncert v okolí lokací, které má přidané, pak při vstupu do sekce lokací mu aplikací doporučí ke sledování další lokace, kde již tito interpreti v dohledné době působí. V sekci koncertů aplikace uživateli zobrazuje plánované koncerty interpretů, na které si může koupit lístek, přidat je ke sledování, či zaškrtnout, že se jich zúčastní. Pokud uživatel zaškrtne pole „interested“ nebo „going“, událost se mu přenese do sekce plány, kde má zobrazené všechny takto označené události a může tak sledovat blížící se události. SongKick lze stáhnout i jako mobilní aplikaci na mobily s operačním systémem Apple či Android. Aplikace je dostupná na odkazu https://www.songkick.com/home.

1.2.2 Ostatní aplikace

Mezi ostatní podobné aplikace na vyhledávání koncertů patří dále takové, které příliš nedisponují dynamičností, co se týče vyhledávání koncertů. Aplikace zahrnují seznam koncertů a událostí jednotlivých interpretů, v nichž může uživatel filtrovat podle základních filtrů. Takové aplikace nevyžadují po uživateli registraci a vyplněný profil, aby mohl vyhledávat koncerty. Disponují pouze základním filtrem, díky kterému si uživatel vyfiltruje, co potřebuje. Aplikace založené na rozšířeném doporučení, které lze označit za doporučovací systém, pak již registraci vyžadují a umožňují uživateli například sledovat událost různých interpretů, které sleduje a zároveň mu doporučuje další interprety podobných žánrů či takových, které sledují i ostatní fanoušci stejných interpretů.

Takovými aplikacemi je například Bandsintown Concerts, Eventbrite, TicketPortal a mnoho dalších.

2 DOPORUČOVACÍ SYSTÉMY

Doporučovací systémy jsou takové systémy, které uživateli umožní zjednodušit výběr jakéhokoliv zboží, produktu, výletu, dovolené a vlastně čehokoliv, na co si uživatel vzpomene (dále jen položka), na základě informací, které má v sobě uložené nebo které ví o daném uživateli. Pomocí těchto informací filtruje výběr položek, které pak uživatel v konečném důsledku nabídne.

2.1 Skupiny doporučovacích systémů

Dle způsobu doporučování se dělí tyto systémy na několik skupin.

2.1.1 Kolektivní filtrování (Collaborative recommendation)

Tento typ doporučení je založen na základní myšlence, že způsob doporučení závisí na sdílení toho, o co se uživatelé v minulosti zajímali. Tedy pokud se uživatelé o něco zajímali v minulosti, je pravděpodobné, že se o to budou zajímat i v budoucnosti. Předpokladem úspěšné predikce je pouze to, aby se nezměnil zájem uživatele. Systém nemá k dispozici žádné podrobnosti o profilu uživatelů či o doporučované položce. Pouze ví, že uživatel si tuto položku vybral na základě hledání a nabídne ji i druhému uživateli, který zadal stejné vyhledávací parametry.

Lze si to představit na jednoduchém příkladu půjčovny knih. Pokud uživatelé A a B mají stejný zájem si zapůjčit knihu o historii a uživatel A si v minulosti vypůjčil danou knihu, kterou uživatel B předtím neviděl, je základním předpokladem systému tohoto typu navrhnout danou knihu uživateli B. Název tohoto doporučování je odvozen od implicitní spolupráce uživatelů, kdy jejich výběry navzájem ovlivňují doporučení pro ostatní uživatele. Dalším příkladem může být e-shop, kde si uživatel rozklikne daný produkt a pod produktem je zobrazena sekce „Ostatní uživatelé také zakoupili“.

Rozdělení kolektivního filtrování

Kolektivní filtrování se dělí na 2 hlavní skupiny. Jedná se o doporučení založené na podobnosti uživatelů a doporučení založené na podobnosti doporučovaných položek. První metoda spočívá v tom, že systém hledá uživateli co nejpodobnější uživatele, kteří mají stejné preference a nabízí mu stejné položky, o které se v minulosti zajímali tito uživatelé. Druhý způsob je naopak založen na tom, že pokud si uživatel v minulosti vyhledal konkrétní položku, je pravděpodobné, že bude vyhledávat jí podobnou položku, tedy mu systém nabízí položky, které jsou si navzájem podobné. V obou případech však systém doporučuje takové položky, se kterýma uživatel do styku ještě nepřišel.

Zdroje [1],[2]

2.1.2 Filtrování na základě obsahu (Content-based recommendation)

Zatímco kolektivní filtrování je definováno jako „Doporučené tituly, které se líbily podobným uživatelům“, filtrování na základě obsahu může být definované, mimo jiné, jako „Doporučené tituly, které se uživateli líbily v minulosti“.

Jádrem obsahově založeného doporučování je dostupnost detailnějšího popisu položek a profilu uživatele, který přiřazuje význam těmto charakteristikám. Jinými slovy položka disponuje několika charakteristikami a uživatel má konkrétní požadavky na doporučovanou položku. Čím více charakteristik odpovídá požadavkům daného uživatele, tím je doporučování přesnější a daná položka má mnohem větší význam pro uživatele než taková, která se shoduje například pouze s jedinou charakteristikou.

Výhodou této metody je, že není potřeba jako v případě kolektivního doporučování velké uživatelské základny. Může spolehlivě doporučit položku i když je v systému pouze jediný uživatel.

Rozdělení filtrování na základě obsahu

I tento systém lze rozdělit na více typů. Rozdělení může spočívat v odvození uživatelského profilu. Ten lze automaticky odvodit například jeho chováním v systému či zpětnou vazbou k daným položkám. Systém se může naučit díky historii uživatelských akcí, že nemá rád daný žánr či daného autora, protože mu přisuzoval nízké hodnocení. Tyto informace poté porovnává vzájemně s ostatními položkami (Může se jednat například o překrytí klíčových slov v knize) a v případě (ne)shody rozhodne o jeho (ne)doporučení. Dalším způsobem je přímé dotazování na uživatelovy preference (například požadované vyplnění formuláře ohledně profilu v systému) a na základě těchto preferencí hledá shody s charakteristikami položek.

Zdroje [1],[2]

2.1.3 Filtrování na základě zkušeností (Knowledge-based recommendation)

Předpokladem pro předchozí dvě metody je existence ukládání historie o vyhledávaných (vypůjčených apod.) položkách. Systémy filtrující na základě zkušeností mají většinou jednorázové uživatele, kteří se do takového systému podívají jednou za čas, a proto není vhodné si o nich ukládat jakékoliv statistiky či historii o jejich akcích v systému. Lze to ukázat na příkladu prodeje elektroniky, například ledničky. Tu si uživatel koupí jednou za několik let, a proto není pro systém vhodné si vytvářet uživatelský profil, či navrhovat ledničku, kterou si oblíbili jiní, což by vedlo k tomu, navrhovat jen špičkové zboží. Systém tedy musí využívat dodatečné informace jak o uživateli (tím, že zjistí jeho požadavky na položku), tak o dostupných produktech (informace o nich má v uložené například v databázi.

Rozdělení filtrování na základě zkušeností

Doporučování na základě zkušeností se dělí na 2 způsoby. Jedná se o doporučení na základě požadavků (case-based) a doporučení na základě omezení (constraint-based). Prvním z nich je způsob, kdy se systém uživatel specifikuje své požadavky a pomocí těchto požadavků systém vyhodnotí doporučované položky na základě podobnosti s těmito požadavky. Soustředí se na nalezení co nejvíce položek, které splňují tyto požadavky. Druhým způsob spočívá v tom, že jsou definovány charakteristiky položky a minimální či maximální požadavky uživatele na dané charakteristiky. Toto omezení lze ukázat například na nákupu kola. Uživatel nechce, aby kolo při ceně nad 30 000 bylo osazené nějakou horší komponentou, když ostatní komponenty jsou prvotřídní. Naopak může definovat, že pokud to kolo bude stát do nějaké nižší částky, tak tyto horší komponenty obsahovat může.

Zdroje [1],[2]

2.1.4 Hybridní systémy

Výše zmíněné systémy mají své silné stránky, ale zároveň i spoustu nedostatků. Tyto nedostatky řeší tzv. hybridní systémy. Jedná se o systémy, které kombinují více přístupů současně, čímž podtrhávají silné stránky těchto systémů a zároveň minimalizují jejich slabé stránky. Spojením více systémů do jednoho hybridního vznikne mnohem přesnější výstup doporučených položek nebo jsou eliminovány nedostatky spojením s vlastnostmi a funkcí druhého systému. Například problém s malým počtem uživatelů v systému založeném na kolektivním filtrování, se kterým se tento systém potýká na začátku, a který vede k nemožnosti určit podobnost mezi uživateli lze vyřešit kombinací se systémem na základě obsahu.

Zdroje [1],[2]

2.2 Doporučování na základě obsahu

V této kapitole bude podrobně rozebrána metoda doporučování na základě obsahu, protože je to metoda, kterou webová aplikace využívá. Jak již bylo napsáno v přehledu výše, doporučování na základě obsahu se dělí na několik typů, které jsou popsány dále v textu.

2.2.1 Reprezentace obsahu a podobnost obsahu

Tato metoda spočívá v hledání shodných charakteristik mezi uživatelským profilem a charakteristikami položek dostupných v databázi. Na příkladu dále je tato metoda popsána. Tabulka 2.1 obsahuje seznam knih v databázi včetně základních charakteristik.

Tabulka 2.1: Seznam knih v databázi

Titul	Žánr	Autor	Cena (Kč)	Typ	Klíčová slova
The night of the gun	Monografie	David Carr	350,-	Brožovaná vazba	tisk a žurnalistika, drogová závislost, osobní vzpomínky, New York
Věštění z krajky	Beletrie, tajemství	Brunonia Barry	360,-	Pevná vazba	Americká moderní fikce, detektivní, historické
Into the fire	Romantika, napětí	Suzanne Brockmann	170,-	Pevná vazba	Americká fikce, vražda, neonacismus

Tyto charakteristiky jsou porovnávány s vyplněným profilem uživatele v tabulce 2.2. Systém může vytvořit profil uživatele několika způsoby. Buď přímým dotazováním na různé charakteristiky (žánr, autora, klíčová slova apod.) nebo nechá uživatele vyplnit profil jen částečně a na základě jeho chování v systému (označení zajímavých knih, zakoupení knih, odmítnutí dříve nabízené knihy apod.) a znalosti svých položek může vyplnit některé charakteristiky (průměrná cena, klíčová slova) sám.

Tabulka 2.2: Preference uživatele

Titul	Žánr	Autor	Cena	Typ	Klíčová slova
…	Beletrie, napětí	Brunonia Barry, Stephen King	400	Pevná vazba, brožovaná vazba	New York, vraždy, detektivní

Na základě vytvořeného profilu poté prochází databázi knih a vyhledává tituly, které odpovídají preferovaným charakteristikám. Těmto titulům poté přidává „váhu zájmu“ uživatele podle množství shodných charakteristik a na základě této váhy mu zobrazí seznam nabízených titulů. Poté už je na uživateli, jak k doporučeným titulům přistoupí. Systém jeho kroky zaznamená a v další relaci s uživatelem je využije pro lepší doporučení.

Zdroj [1],[2]

2.2.2 Vyhledávání na základě podobnosti

Jedny z metod využívající tento typ vyhledávání jsou metoda „Nejbližšího souseda“ či Rocchiova metoda „Relevantní zpětné vazby“.

Nejbližší soused

Pro realizaci tohoto způsobu vyhledávání jsou nutné 2 podmínky. První je mít k dispozici historii charakteristik „líbí“/“nelíbí“ určených uživatelem. To je možné získat přímým dotázáním nebo monitorováním chování uživatele v systému. Druhá podmínka je mít k dispozici nástroj, který určí podobnost dvou položek. Při splnění těchto podmínek lze tento způsob doporučování využít.

Pro příklad lze opět využít systém prodeje knih. Systém by měl nabídnout uživateli knihu, kterou ještě neviděl. Pokud například 4 z 5 nejpodobnějších knih k této dané knize uživatel v minulosti označil, že se mu líbily, je velmi vysoká pravděpodobnost, že i kniha titul se mu bude líbit, a tak je pro systém žádoucí jej uživateli nabídnout.

Takovýto systém se ještě rozděluje do tzv. „krátkodobého“ a „dlouhodobého“ modelu. Rozdíl mezi těmito modely je, že pokud systém pracuje v krátkodobém modelu, hledá podobné položky na základě nedávno ohodnocených. To je pro uživatele žádoucí, když například změní charakteristiku, či přidá novou. Například si uživatel přečetl novou sci-fi knihu a zalíbil se mu tento žánr, tak si chce přečíst další knihy stejného žánru. Naproti tomu dlouhodobý model spravuje uživatelovi kroky v delším měřítku. Pro příklad, uživateli se sice zalíbil nový žánr, ale pořád se mu líbí i ty starší, které si oblíbil dříve v minulosti, a nejen pouze v poslední době.

Pro systém je největší problém, jaký z těchto dvou modelů zvolit, aby byl uživatel co nejvíce spokojen s výsledkem doporučení. Možností kombinace těchto modelů je několik. Jedním z nich může být, že systém vyhledá nejdříve v krátkodobém modelu a pokud zde nalezne málo výsledků nebo žádný, tak přejde do dlouhodobého modelu.

Relevantní zpětná vazba

Metoda relevantní zpětné vazby je založena na trochu odlišném způsobu vyhledávání dle obsahu. Jako předchozí metoda nevyužívá informace o preferencích, ale místo toho spoléhá na uživatelovu zpětnou vazbu o položkách, což systému řekne, jestli pro něj byly relevantní nebo ne. Tato informace je pak použita pro další výsledky. Představit si to lze na e-shopu s elektronikou. Uživatel si zakoupí položku a systém se ho poté zeptá, jestli se mu daná položka líbila nebo ne. Na základě této informace mu znovu podobné položky (ne)nabízí.

Na rozdíl od předchozí metody není tento způsob doporučování omezený na individuálních uživatelských preferencích, dle kterých se poté vyhledávají položky, ale závisí čistě na mechanismu zpětné vazby a tomu, jak je systém naimplementovaný. Dobře naimplementovaný systém pak ze zpětné vazby uživatele a mechanismu pro hledání podobných položek dokáže velmi přesně doporučit další relevantní položky pro uživatele.

Zdroje [1],[2]

2.3 Shrnutí

Doporučovací systémy nabízejí techniky, které využívá spousta organizací, e-shopů, půjčoven apod. k tomu, aby co nejvíce uspokojili zákazníka, či uživatele a nabídli mu tak přesně to, co očekává, čímž mu zjednodušili výběr, či mu rozšířili obzory.

3 POUŽITÉ TECHNOLOGIE

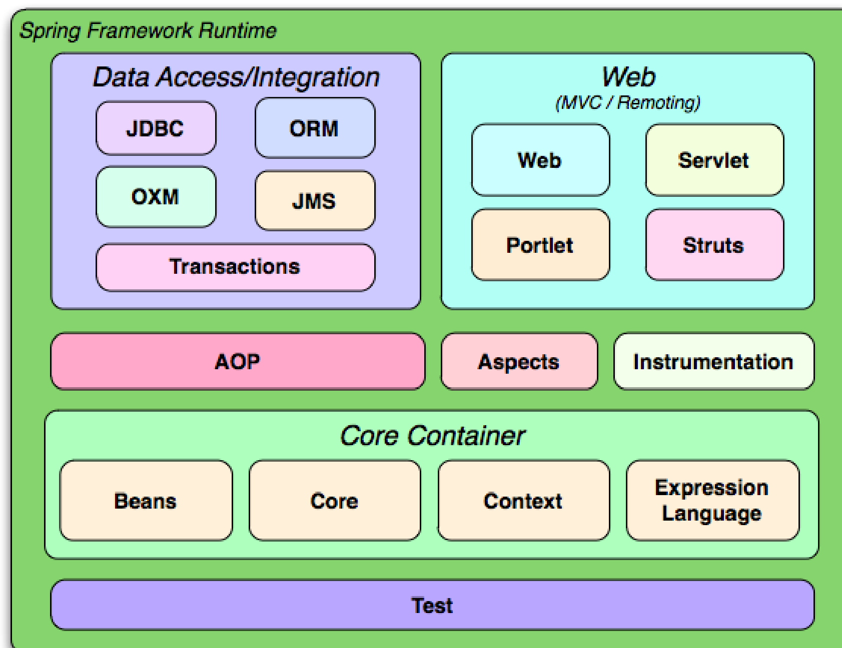
V rámci aplikace bylo využito několik technologií. Celá aplikace je rozdělena do dvou částí – backend a frontend. Backendová část je napsána v programovacím jazyku Java s využitím Spring frameworku, resp. jeho zjednodušené verze SpringBoot. Frontendová část je napsaná v JavaScriptovém frameworku React.js. Aplikace využívá databázi MySQL.

3.1 Spring framework

Spring framework je open-source platforma, která poskytuje komplexní infrastrukturu pro vývoj tzv. „Enterprise Java aplikací“. Spring je vysoce modulární, což vývojáři umožňuje využívat pouze ty části, které potřebuje v rámci vyvíjené aplikace, aniž by musel zahrnovat do projektu všechny části, které Spring nabízí. Všechny jeho části jsou od sebe navzájem odděleny, aby nebyly závislé jedna na druhé. Stejný přístup pak přináší i do samotné aplikace. V integrační vrstvě aplikace budou existovat různé návaznosti na Spring, nicméně by mělo být snadné tyto návaznosti izolovat od zbytku kódu.

Spring jako takový nepřináší nové technologie. Pouze vzal stávající, plně funkční technologie a integroval je do sebe tak, aby je vývojář mohl společně lépe využít při vývoji aplikací. Celý Spring framework je rozdělen do přibližně dvaceti modulů. Tyto moduly jsou rozděleny do několika vrstev – kontejner jádra, web, integrace do databáze, AOP, instrumentace a test, jak je vidět z obrázku 3.1.

Zdroje [3],[4]



Obrázek 3.1: Rozdělení Spring framework modulů. Zdroj: převzato z [4]

3.1.1 Dependency injection

Jedním z nejdůležitějších přínosů Springu je tzv. dependency injection („vkládání závislostí“), který je součástí principu známém jako „Inversion of Control“ (inverze řízení – princip, kterým je řízení objektů či částí programu přeneseno na framework). Třídy Java aplikace by měly být na sobě co nejvíce nezávislé hlavně z důvodu znovupoužitelnosti těchto tříd a také pro nezávislé testování jednotlivých tříd. Principem této techniky je vložení závislosti třídy A do třídy B tak, aniž by třída B byla zodpovědná za celý životní cyklus třídy A. O ten se stará výhradně kontejner, v tomto případě kontejner Spring frameworku. Od kontejneru poté třída B vyžaduje pouze referenci na danou třídu, kterých může být hned několik, což dělá DI tak silným nástrojem. Je například možné využít jinou třídu pro vývojářský režim, testování a pro produkci.

Zdroje [3],[4]

3.1.2 Aspektově-orientované programování

Dalším z klíčových komponent Springu je AOP neboli aspektově orientované programování. Aspekt je v tomto případě jednotka modularity aplikace. Existují různé funkce, které jsou používány v průřezu celé aplikace. Tyto funkce jsou tzv „cross-cutting“ koncerny neboli „průřezové koncerny“ a jsou koncepčně odděleny od logiky aplikace. Proto by měly být nezávislé v rámci aplikace. Patří mezi ně například logování, auditování či zabezpečení. AOP pomáhá oddělit tyto aspekty od objektů, které ovlivňují. V rámci aplikace je implementováno logování, zabezpečení a ukládání do mezipaměti.

Zdroje [3],[4]

3.1.3 Beans

Další z důležitých součástí Springu jsou nazývané „beany“. Jsou to páteřní objekty Spring aplikace. Beana je objekt, o jehož celý životní cyklus se stará IoC kontejner. Kontejneru jsou beany dodány s konfiguračními metadaty. Beany mohou být definovány třemi způsoby. Pomocí XML konfiguračního souboru, pomocí anotací či na základě Java příkazů. Aby mohl kontejner pracovat s beanami musí vědět, jak vytvořit beanu, jaký je její životní cyklus a její závislosti. Právě beany jsou pak objekty, které DI vkládá do ostatních objektů, jak je uvedeno v kapitole 3.1.1.

Zdroje [3],[4]

3.1.4 SpringBoot

Aplikace v rámci této aplikace je zpracována pomocí SpringBootu. SpringBoot vznikl jako „odlehčená“ verze Springu, která by měla urychlit vývoj aplikací, které nejsou příliš robustní. Při vývoji aplikace ve Springu je proces konfigurace jednotlivých bean, modulů, nastavení různých zabezpečení apod. velmi náročný a zdlouhavý. To je pro programátora, který není ve Springu příliš zběhlý, nepříjemná záležitost. Z tohoto důvodu vznikl projekt SpringBoot, který programátorovi urychlí základní nastavení aplikace. Aplikaci je nutné přidat anotaci @SpringBootApplication, aby bylo jasné, že se jedná o SpringBoot aplikaci. Tato anotace v sobě zahrnuje další 3 anotace:

- @*Configuration* – anotace značí, že se jedná o konfigurační třídu. V této třídě mohou být registrovány další extra beany v kontextu, importovány nebo implementovány další konfigurace,
- @EnableAutoConfiguration – anotace zajišťující automatickou konfiguraci modulů, které ji vyžadují (např. modul *spring-boot-starter-data-jpa),*
- @ComponentScan – umožní sken komponent ve složkách (v Javě balíčky) ve stromové struktuře na stejné nebo nižší úrovni.

Server

SpringBoot má v sobě zabudovaný Tomcat server, který se ve výchozím stavu automaticky zapne při spuštění aplikace (pokud není nastaveno jinak). Pokud programátor chce využít jiný server, tak provede potřebné nastavení, zabudovaný Tomcat server se potlačí a je využit vybraný server s potřebnou konfigurací přesně na míru aplikace.

Nastavení

Při vývoji SpringBoot aplikace je důležitý soubor *application.properties*. Některé naimportované SpringBoot moduly (vizte kapitola 3.1.5 Maven) vyžadují základní nastavení, které se nachází právě v tomto souboru. Zde jsou definovány proměnné povinné pro některé moduly. Například při importování modulu *spring-boot-starter-data-jpa* je nutné nastavit proměnné pro databázi, vizte ukázka kódu 3.1.

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
spring.jpa.show-sql=true
spring.datasource.url=jdbc:mysql://localhost:3306/musicfans-app?useSSL=false
spring.datasource.username= username
spring.datasource.password=password
```

Ukázka kódu 3.1: Ukázka nastavení pro databázi v souboru application.properties

Programátor si do tohoto souboru může nastavit jakékoliv proměnné, které v rámci projektu potřebuje a mění se například v závislosti na tom, jestli je aplikace použita v testovacím či produkčním prostředí.

Pro nastavení třídy, která má být beanou je zase nutné pouze označit tuto třídu anotací (např. @*Controller*, @*Service*, @*Component*) a SpringBoot se zase postará o zbytek. Programátor tedy nemusí provádět zdlouhavé nastavení v konfiguračním XML souboru s metadaty.

Zdroj [5]

Spring vs Springboot

Spring i samotný SpringBoot mají svá pro a proti. SpringBoot umožňuje rychlejší vývoj aplikací, a proto se hodí spíše na aplikace, které nejsou příliš robustní a na jejich vývoj je například vyhrazeno málo času. Vývojář má během pár minut k dispozici funkční aplikaci, včetně potřebné konfigurace. Ve spoustě případů ovšem neví, co se děje v pozadí při spouštění aplikace (například při konfiguraci) a musí se na SpringBoot spoléhat. Pokud mu nestačí výchozí nastavení, o které se postará SpringBoot, tak se musí do samotné konfigurace pustit sám, což už také mnohdy vyžaduje znalosti celého Spring frameworku. Při vývoji velmi rozsáhlé aplikace, která vyžaduje mnoho modulů už je potřeba i zkušený vývojář, který je se Springem velmi dobře obeznámen a ten raději využije samotný Spring. Veškeré nastavení si provede sám, a tak přesně ví, co a jak má nastaveno a co se kde děje. To všechno zase na úkor zdlouhavé konfigurace a mnoho konfiguračních souborů navíc.

3.1.5 Manažer pro build aplikace

Pro (nejen) importování modulů *Springframework* používá nástroj Maven nebo Gradle. Oba nástroje jsou softwary pro správu a konfiguraci projektu a umožňují tak importovat různé knihovny, které jsou v rámci projektu. Zároveň se starají například o automatizace procesů, nastavení různých pluginů, build aplikace apod. Maven je založen na XML kontextu, naopak

Gradle je založen na kontextu Groovy. Využití jednoho nebo druhého nástroje je čistě na vývojáři. V rámci aplikace je využit Maven. Jeden z hlavních přínosů je však v importování jednotlivých modulů (knihoven). Pokud chce vývojář využít knihovny, které nejsou obsaženy v základním Java SDK musí je v klasickém případě ručně stáhnout a importovat do svého projektu. Pro velkou aplikaci je to zdlouhavá a náročná operace. Maven umožňuje tento proces výrazně urychlit. Vývojáři stačí pouze do části se závislostmi (tzv. „dependencies“) pro jednotlivou závislost nastavit tzv. „groupId“ (např. *org.springframework.boot*), „artifactId“ (např. *spring-boot-starter-data-jpa)* a „version“ (např. *2.1.4.RELEASE*), případně pak další nastavení, vizte ukázka kódu 3.2. Maven se postará o stažení těchto knihoven a automatický import do projektu. Pokud se například vymění knihovna a stará již nebude potřeba, tak se jednoduše odstraní z konfiguračního souboru a Maven se postará o její úplné odstranění z projektu. Zároveň se tímto zjednodušuje týmová spolupráce na projektu, protože se ostatní programátoři nemusí o tyto změny starat sami.

Zdroj [6]

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.6</version>
    <scope>provided</scope>
</dependency>
```

Ukázka kódu 3.2: Ukázka importu knihovny pomocí Maven

3.2 REST

REST je architektura, pomocí níž lze zpracovávat data na serveru. Poprvé byl představen v roce 2000 v disertační práci Roy Thomas Fieldinga, který je mimo jiné jedním z hlavních autorů HTTP, což je aplikační protokol, který je základem komunikace po internetu. Právě na tomto protokolu je REST postaven, neboť využívá některé z jeho metod ke svým vlastním účelům. Jedná se o metody GET (čtení dat), POST (zápis dat), PUT (aktualizace dat) a DELETE (mazání dat).

Dle Fieldinga není nutné přesně dodržovat principy metod a jejich primární využití. Například pro získání dat lze využít metodu POST. POST lze použít i na aktualizaci či mazání dat. Pro zápis dat lze naopak využít metodu GET. S tím ale už souvisí i některá rizika ohledně bezpečnosti celé aplikace, a proto by se měly používat metody k účelům, pro které byly určeny.

Podobně jako ostatní architektonické styly i REST musí splňovat základní podmínky, aby aplikační rozhraní mohlo být označeno jako RESTfull. Tyto podmínky jsou podrobněji popsány v následujících kapitolách.

Zdroj [7]

3.2.1 Klient-server

První a zároveň nejdůležitější podmínkou je využití architektonického stylu klient-server. Oddělením klienta od webového rozhraní lze zvýšit přenositelnost uživatelského rozhraní napříč platformami a zároveň zlepšit škálovatelnost zjednodušením serverových komponent. Lze mít například jako klienta web a zároveň mobilní aplikaci, kdy obě platformy využívají jeden server.

Zdroje [7],[8]

3.2.2 Bezstavovost

Dalším důležitým omezením je bezstavová komunikace mezi klientem a serverem. Server by měl vždy z požadavku klienta pochopit, co po něm klient vyžaduje. Proto by měl požadavek obsahovat všechny potřebné informace, aby byl zcela nezávislý na kontextech uložených na straně serveru. Stav relace je celý uchováván na straně klienta.

Zdroje [7],[8]

3.2.3 Využití mezipaměti

Jedním z dalších požadavků je možnost označit data z odpovědí za tzv „cacheable“ (možnost využít mezipaměť) nebo tzv „non-cacheable“ (nemožnost využít mezipaměť). V případě povolení mezipaměti na odpovědi ze serveru je zvýšena efektivita celého systému, protože klient má právo znovu použít data odpovědí ze serveru pro pozdější požadavky, které již byly v rámci relace na server volány.

Zdroje [7],[8]

3.2.4 Jednotné rozhraní

Důraz na jednotné rozhraní komponent je hlavním rysem, který odlišuje architektonický styl REST od ostatních webově založených stylů. Použitím principu softwarového inženýrství, který je obecný na rozhraní komponent, je zjednodušena celková architektura systému a zlepšuje se viditelnost interakcí.

Zdroje [7],[8]

3.2.5 Vrstvený systém

Tím, že je komponentám umožněno „vidět“ pouze bezprostřední vrstvu, se kterou interaguje umožňuje celé architektuře styl vrstveného systému skládat se z hierarchických vrstev. Tím, že se omezí znalosti systému na jednu vrstvu jsme vázáni na celkovou složitost systému a je podpořena nezávislost služeb v rámci vrstvy. Jednotlivé vrstvy mohou být použity pro zapouzdření starších služeb a zároveň pro ochranu nových služeb před staršími klienty.

Zdroje [7],[8]

3.2.6 Kód na vyžádání

REST umožňuje rozšířit funkčnost klienta stažením s puštěním kódu ve formě appletů nebo skriptů. Tento přístup zjednodušuje klienty tím, že redukují počet funkcí, které mají být předem implementovány.

Zdroje [7],[8]

3.2.7 XML vs JSON

Nejčastějším formátem, kterým jsou přenášena data v těle http zprávy skrze REST je formát XML nebo JSON. Webovou službu lze nastavit tak, aby akceptovala tzv „MIME typ“ pro jeden z těchto formátů. Tabulka 3.1 zobrazuje možné zapsání těchto typů v hlavičce http požadavků.

Tabulka 3.1: Možnost zápisu typu těla HTTP zprávy

Formát HTTP	Content-type
JSON	Application/json
XML	Application/xml

Různý formát přenášených HTTP zpráv zvyšuje univerzálnost komunikace mezi webovou službou a klienty, kteří ji využívají. Ti mohou být napsaní jakýmkoliv programovacím jazykem, který umožňuje tento formát zpracovat a mohou běžet na různých platformách či zařízeních.

3.3 React

React je JavaScriptová knihovna pro vytváření uživatelských rozhraní vyvíjená společností Facebook. React, podobně jako mnohé moderní JS knihovny je založen na „single-page" architektuře. Ta spočívá v tom, že je načtena jedna HTML stránka a obsah stránky se dynamicky mění podle toho, jak se uživatel na stránce pohybuje. Při změně cesty v URL adrese (v rámci pohybu v aplikaci, nikoliv při ručním zásahu do URL) nedojde k aktualizaci, a tedy překreslení celé stránky, ale pouze se upraví částečný obsah této jedné stránky.

Zdroj [9]

3.3.1 Virtual DOM

React využívá Virtual DOM, který se používá pro selektivní překreslování uživatelského rozhraní (tzv. „re-rendering"), které jej činí velmi efektivním. Virtuální DOM je lehká kopie skutečného DOM, kterým disponuje webová stránka. Oproti skutečnému DOM je mnohem rychlejší. Po aktualizaci virtuálního DOM je porovnáván snímek (tzv. „snapshot") s předchozím snímkem virtuálního DOM. Po srovnání React ví, které části byly změněny a ty pak převede do skutečného DOM, čímž uživateli aktualizuje celou stránku.

Zdroj [10]

3.3.2 JSX

JSX (Javascript syntax extension) je rozšíření syntaxe pro Javascript. Není povinné v rámci React aplikace využívat JSX, ale má svoje výhody, které dělají vývoj React aplikace jednodušším. Nejužitečnější nástroj, který JSX nabízí je transformace klasického formátu HTML do objektu JavaScriptu a vložit tak výraz JavaScriptu do HTML kódu pomocí složených závorek, čímž pak vznikne React element, jak je vidět v ukázce kódu 3.3. Další užitečnou funkcí je ochrana před tzv „inject" útoky. JSX vždy před vykreslením všechno převede do textu a není tak možné upravit nějaké hodnoty.

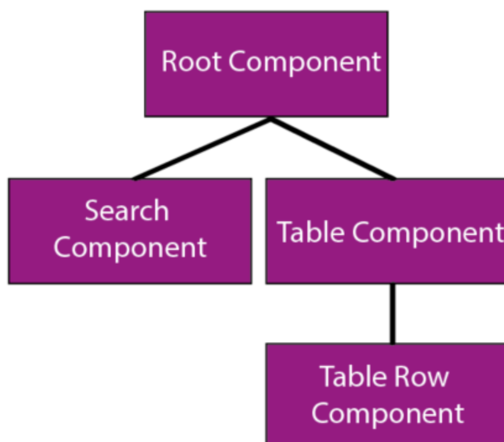
Zdroj [10]

```
class Hello extends React.Component {
    render() {
      return <h1Hello World {this.props.user}
    }
}
```

Ukázka kódu 3.3: Ukázka funkce JSX

3.3.3 React komponenty

React je založený na komponentách. Tyto komponenty jsou nezávislé a znovupoužitelné. Celá stránka se pak skládá s jednotlivých komponent tvořících stromovou strukturu. Komponenty mohou být pak například poskládány, jak je vidět na obrázku 3.2. Tok dat jde vždy od kořenové komponenty a končí u tzv. „listů" celého stromového modelu.



Obrázek 3.2: Ukázka stromové struktury. Zdroj: autor

Props a State

Props a *state* jsou vstupní data pro vykreslování komponent. Jedná se o JavaScriptové objekty a jejich změna zapříčiní překreslení komponenty (pokud není uvedeno jinak). Jednotlivé komponenty mohou být stavové i bezstavové. Stavové komponenty si drží svůj stav v objektu *state*, který lze libovolně měnit. To je možné pouze pomocí metody *setState()*, což zajistí update komponenty. V ukázce kódu 3.4 je uveden příklad, jak změnit *state*.

```
this.state.name = „Radim" – chyba, takto stav měnit nelze a bude na to upozorněno
this.setState({name: „Radim"}) – hodnota „name" v objektu State je změněna a zároveň se
zajistí aktualizace komponenty
```

Ukázka kódu 3.4: Ukázka změny stavu komponenty

Props, na rozdíl od *state* je neměnný, takže komponenta jej nemůže změnit. *Props* jsou do komponenty jsou vloženy pomocí předka, který jediný je může změnit. Bezstavové komponenty objektem *state* nedisponují a svůj stav si tak řídit nemohou. V Ukázce kódu 3.5 je zobrazeno vytvoření jednoduché komponenty. Komponenta si ve svém objektu *state* drží proměnnou *greeting* s hodnotou „Hello“. Tu může libovolně pomocí metod měnit. Od svého předka získala v objektu *props* proměnnou *name*, kterou měnit nemůže. O vykreslení se stará povinná metoda *render()*, která v tomto případě vypíše uživateli nadpis: „Hello Radim“.

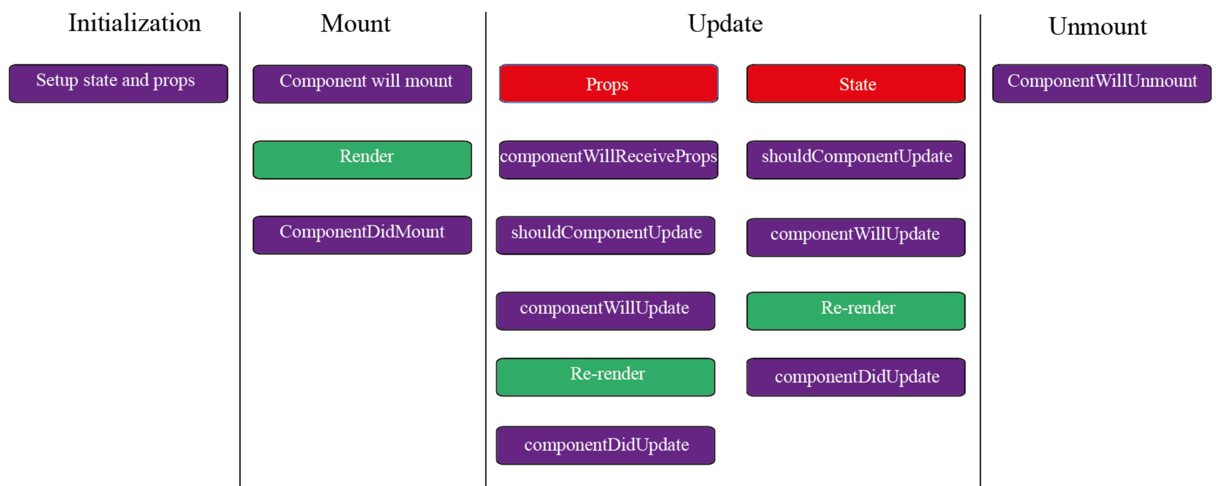
```
export default class Hello extends React.Component {
        constructor(props){
                this.state = {
                        greeting: “Hello”
        }
    render() {
        return <h1>{this.state.greeting} {this.props.name}</h1>;
    }
}
Použití komponenty: <Hello name=“Radim“/>
```

Ukázka kódu 3.5: Ukázka vytvoření a použití komponenty

Ve vykreslovací metodě *render()* musí všechny komponenty a elementy vždy náležet pouze do jednoho kořenového elementu. V ukázce výše je pouze element „h1“, takže je vše v pořádku. V případě, že pod sebou budou elementy dva, bylo by nutné je obalit například do jednoho elementu „div“. Zdroje [9], [10]

3.3.4 Životní cyklus komponenty

Všechny React komponenty mají svůj vlastní životní cyklus, který je rozložen do 4 částí – inicializace, počátek, aktualizace a zánik. V rámci těchto cyklů se pak provedou předdefinované metody, které vývojář může implementovat podle potřeby. Stejně tak je nemusí implementovat vůbec, pokud je v dané komponentě nepotřebuje. Na obrázku 3.3 je zobrazen přehledný graf celého životního cyklu komponent.



Obrázek 3.3: Graf životního cyklu komponenty. Zdroj: autor

Inicializace (initialization)

V této části se nastaví vlastnosti *props* a *state*.

Počátek (mount)

V tomto cyklu se nejdříve provede metoda *componentWillMount*, poté se provede vykreslení, a nakonec se provede metoda *componentDidMount*.

Aktualizace (update)

Tento cyklus je rozdělen do dvou částí. Buď může být aktualizována hodnota v *props* nebo ve *state*. V obou případech se spustí metoda *shouldComponentUpdate*, která ve výchozím stavu zajistí aktualizaci, což spustí metodu *componentWillUpdate*, provede se překreslení a spustí se metoda *componentDidUpdate*. Tento cyklus se opakuje vždy, když se změní stav komponenty či hodnota nějakého atributu ve vlastnosti *props*.

Zánik (unmount)

Před samotným zánikem je spuštěna metoda *componentWillUnmount.*

3.3.5 ES6

ES6 neboli ECMAScript 2015, je standardizovaný skriptovací jazyk a JavaScript je jedna z jeho implementací. React využívá tohoto standardu, který přináší některé novinky a zjednodušení v definici funkcí, zápisu apod.

Zdroj [10]

3.3.6 Funkce

Funkce v JavaScriptu se dají zapsat pomocí klíčového slova „function“. ES6 přináší zápis pomocí tzv. „šipkové notace“. V React je právě tento zápis běžnější a více využívaný. V ukázce kódu 3.6 jsou vidět 2 možné zápisy funkce *hello*, která dostane jeden parametr *greeting*. (V případě dvou a více parametrů musejí být v závorce a oddělené čárkou) a ten vrátí při jejím volání. V prvním případě je nutné klíčové slovo return. U šipkové notace není potřeba klíčového slova „return“. Výraz je vždy implicitně vrácen z funkce.

Zdroj [10]

```
1. function hello (greeting) {return greeting;}
2. const hello = greeting => {greeting}
```

Ukázka kódu 3.6: Ukázka zápisu funkce

3.3.7 Seznamy

Pro práci se seznamy se využívá funkce „map“, která tuto práci zásadně usnadňuje. Místo klasického *for cyklu*, kterým lze projít celé pole, lze využít právě funkci *map* společně se šipkovou notací. V ukázce kódu 3.7 je zobrazen způsob, jakým lze ze seznamu objektů žánrů vytvořit formát, který vyžaduje komponenta na výběr více žánrů. Seznam *genres* obsahuje žánry, které lze vybrat při vyplňování formuláře. React komponenta vyžaduje přesný formát s klíčovými poli *label* a *value*, a proto do seznamu *jsonGenres* jsou tyto žánry namapovány v požadovaném formátu. Tento objekt je pak předán komponentě pro výběr, která k nim přistupuje pomocí vlastnosti *props* a zpracuje je dle toho, jak je implementována. Funkci „map“ je možné kombinovat i s různými filtry, ale i vloženými funkcemi apod. Zápis se tak výrazně zjednoduší.

Zdroj [10]

```
genres.map(value => jsonGenres.push({ label: value.genreType, value: value.idGenre }));
```

Ukázka kódu 3.7: Ukázka použití funkce map

3.3.8 Události

Zachytávání událostí je podobné jako zachytávání událostí na prvcích v DOM. Rozdíl je pouze v tom, že React využívá tzv. *camelCase* notaci (první písmeno je vždy malé a každý další slovo začíná velkým písmenem) pro pojmenování událostí. Pro zamezení výchozího chování

komponenty, na které je událost zachycena nelze vrátit *false*, ale je nutné použít funkci *preventDefault()* volanou nad objektem události *event*, který je vložen automaticky. Ukázka kódu 3.8 ukazuje definování a použití události, která se zavolá po kliknutí na tlačítko a vypíše uživateli zprávu s upozorněním.

Zdroj [10]

```
#definice
handleButtonClicked = (event) =>{
    alert('Button was clicked!');
    event.preventDefault();
}
#použití
<Button value="Click me!" onClick={this.handleButtonClicked}/>
```

Ukázka kódu 3.8: Definice a použití zachycení události po stisku tlačítka

4 ANALÝZA WEBOVÉ APLIKACE

Praktická část webové aplikace je zaměřena na vytvoření webové aplikace, která bude typu klient-server, v technologiích Java EE (Spring Framework) a React.js. Aplikace by měla sloužit všem hudebním fanouškům, kteří vyhledávají hudební události v jejich blízkém okolí, ale i organizátorům, kteří chtějí ve svém okolí hudební událost pořádat, ale neznají priority hudebních fanoušků v okolí. Aplikace bude dodržovat MVC architekturu a bude rozdělena na 2 části. Serverová část (backend) bude REST API, které bude obsahovat logiku aplikace a klientská část aplikace (frontend), se kterou bude uživatel interagovat. Tyto části mezi sebou budou komunikovat pomocí REST rozhraní.

4.1 MVC

MVC je nejpoužívanější architektonický model především při vývoji webových aplikací. Základním principem je oddělení logiky aplikace od výstupu na obrazovku. Celý model je rozdělen do 3 částí – Model, View a Controller. Tyto části by od sebe měly být co nejvíce oddělené.

Model

Model obsahuje celou logiku aplikace a vše, co do ní spadá. Stará se například o náročné výpočty, přístup k databázi a zpracování dotazů a mnoho dalšího. Očekává parametry na vstupu, které zpracuje předem definovaným způsobem a na výstup pošle výsledek zpracování. Nezajímá se vůbec o to, odkud parametry pocházejí ani o to, jak je dále prezentován jeho výstup.

View (pohled)

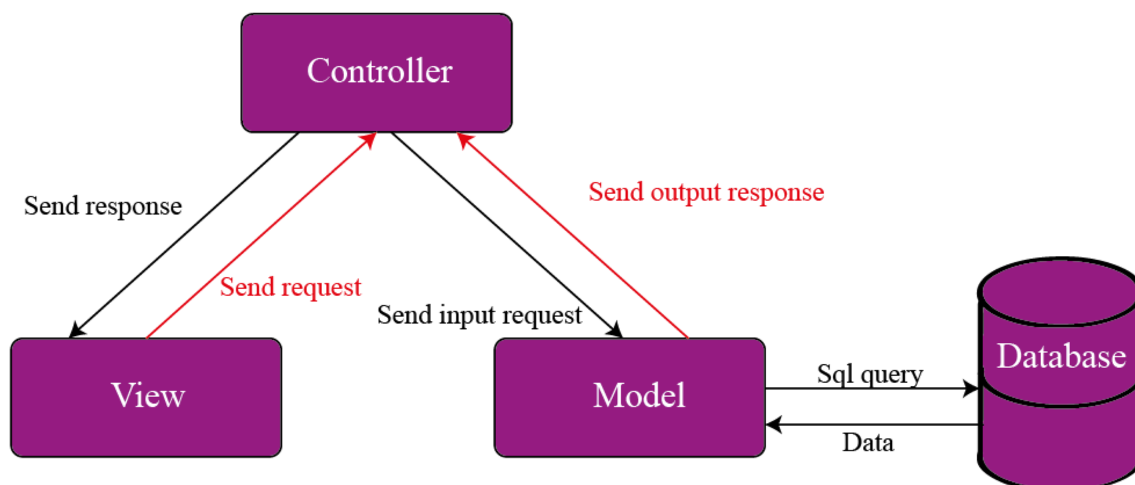
Vrstva View je prezentována uživateli, tedy to je přesně to, co uživatel vidí na obrazovce. Vrstva View by měla obsahovat co nejméně logiky a starat se především a zobrazení výstupních dat z Modelu.

Controller (kontroler)

Kontroler je prostředník mezi vrstvami Model a View. Je to vrstva, která drží celou aplikaci pohromadě. Kontroler komunikuje se všemi vrstvami modelu. Uživatel skrze vrstvu View komunikuje s kontrolerem, který pošle data na vstup vrstvy Model. Ta provede požadované kroky v závislosti na vstupu a pošle výsledek zpracování na výstup. Ten předá kontroler vrstvě View a ta jej opět zobrazí uživateli.

Zdroj [11]

Celý MVC model je zobrazen na grafu v obrázku 4.1



Obrázek 4.1: Graf MVC modelu. Zdroj: autor

4.2 Funkční a nefunkční požadavky

Požadavky na aplikaci lze rozdělit na funkční a nefunkční požadavky. Jako funkční požadavky si lze představit přímé požadavky uživatele na systém, tedy takové, které popisují požadovanou funkci systému. Je to například možnost zobrazení profilu, vyhledání zboží či odhlášení ze systému. Za nefunkční požadavky lze považovat podmínky uvalené na celý systém. Může se jednat o zabezpečení, omezení, výkon, škálovatelnost apod. V kapitole 4.2.1 a 4.2.1 jsou zobrazeny požadavky na doporučovací systém pro hudební fanoušky rozdělené do skupin.

Zdroj [12]

4.2.1 Funkční požadavky

Funkční požadavky se rozdělují do několika skupin, jako jsou požadavky na přístup, funkce aplikace, business logiku apod.

Přístup

- Registrace do systému,
- přihlášení do systému,
- odhlášení ze systému,
- uživatelské role.

Funkce aplikace

- Vyplnění profilu,
- načtení doporučených událostí,
- načtení vlastních událostí,

- načtení událostí doporučených k založení,
- editace vlastních událostí.

Správa

- Zobrazení seznamu interpretů,
- zobrazení seznamu žánrů,
- vložení nového interpreta,
- zobrazení seznamu uživatelů,
- zobrazení seznamu událostí.

Business logika

- Validace formulářů,
- odesílání emailů.

4.2.2 Nefunkční požadavky

Nefunkční požadavky, podobně jako ty funkční, jsou dále rozděleny do skupin bezpečnost, rozšiřitelnost, dostupnost.

Bezpečnost

- Aplikace by měla využívat moderní metody zabezpečení,
- všechna volání přihlášených uživatelů na REST API by měla obsahovat autorizaci,
- aplikace by měla ochraňovat citlivá data uživatelů.

Rozšiřitelnost

- Aplikace by měla být dále rozšiřitelná,
- aplikace by měla umožnit snadné přidání nových funkcionalit.

Dostupnost

- Aplikace by měla být dostupná online pomocí webového prohlížeče.

4.3 Případy užití

Případy užití definují jednotlivé kroky, které definují interakci mezi systémem a uživatelem. Webová aplikace má 3 základní role, které mohou systém využívat. Jsou to organizátor, fanoušek a admin. Každá role má k dispozici různé možnosti využití aplikace. Tyto možnosti definuje tzv. „use case diagram“ neboli diagram případů užití. Diagram je rozdělen do dvou částí. První jsou aktéři. To mohou být jednotlivé role, ale i aplikace třetích stran (např. Google Maps API). Druhá část je hranice systému, která obsahuje případy užití, které jsou přiřazeny

jednotlivým aktérům. Na obrázku 4.2 je zobrazen diagram případů užití pro webovou aplikaci. Ten popisuje jednotlivé aktéry a jim umožněné akce v systému.

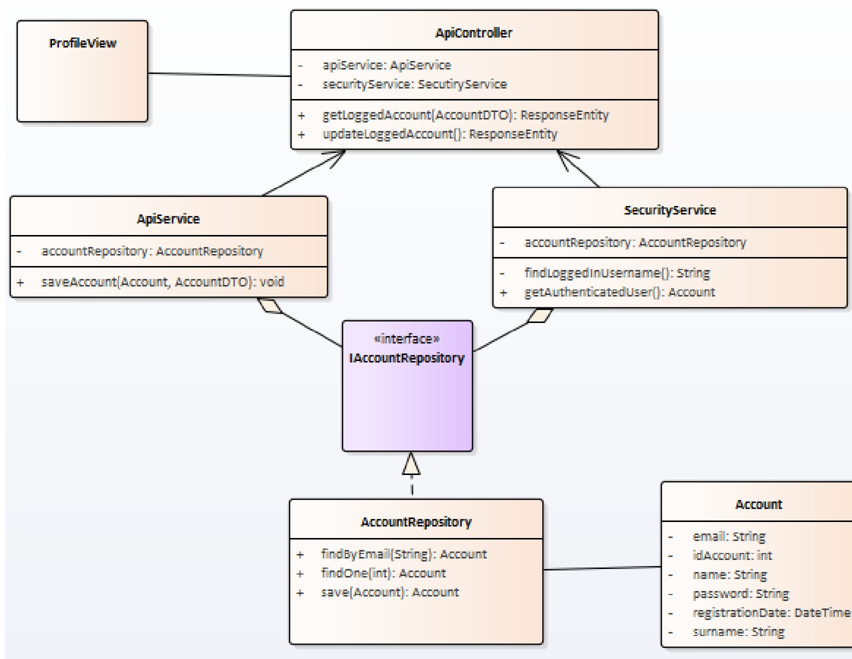
Zdroj [12]

Obrázek 4.2: Diagram případů užití. Zdroj: autor

4.4 UML

UML je grafický jazyk pro návrh a dokumentaci softwaru. Forma jeho zápisu je standardizovaná a nabízí přesně stanovená pravidla pro modelování. Podporuje objektově orientovaný přístup k analýze a návrhu aplikací. Na obrázku 4.3 je zobrazen model, který popisuje architekturu aplikace v případě úpravy uživatelského profilu. K dispozici je třída *ProfileView*, což je vzhled aplikace. Ta komunikuje s kontrolerem, který využívá 2 služby. Jednu pro zabezpečení a druhou pro uložení uživatelského profilu. Obě služby jsou závislé na repozitáři, který komunikuje s databází a spravuje třídu *Account*, která je zároveň tabulkou v databázi. Celý UML model zde zobrazen není, ale ostatní procesy mají podobnou strukturu

Zdroj [12]

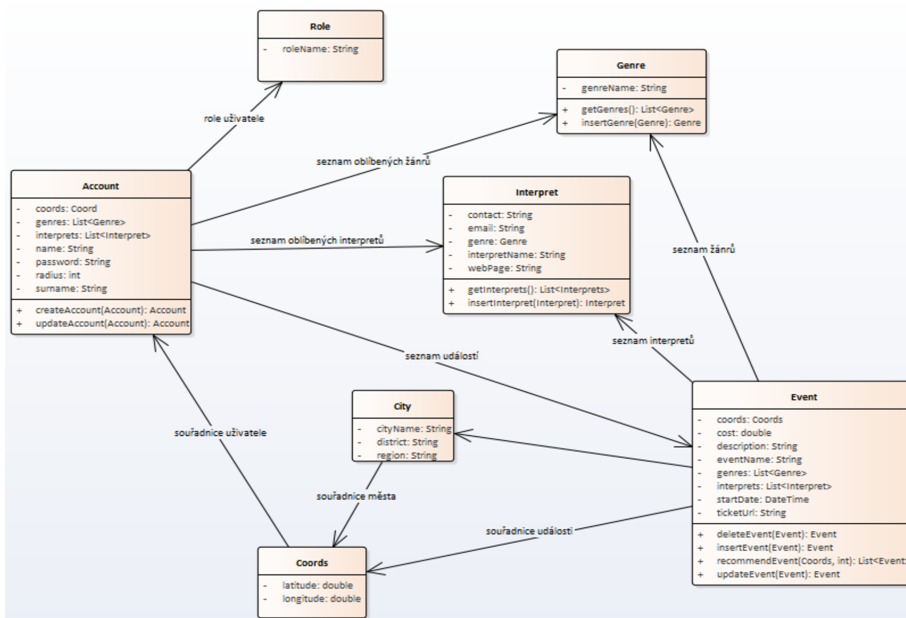


Obrázek 4.3: Ukázka UML – editace profilu. Zdroj: autor

4.5 Model analytických tříd

Analytické třídy popisují abstrakci problémové domény. Lze si to představit jako zjednodušený model toho, jak bude aplikace ve výsledku vypadat. Měly by obsahovat pojmy z reálného světa. Názvy tříd a atributy by proto měly být pečlivě vybrány. Ve spoustě případů model analytických tříd kopíruje datový model nebo se mu hodně přibližuje. Na obrázku 4.4 je vidět model analytických tříd, který přibližně popisuje souvislosti mezi jednotlivými třídami.

Zdroj [12]



Obrázek 4.4: Model analytických tříd. Zdroj: autor

4.6 Datový model

Datový model představuje uspořádání tabulek v databázi a jejich vzájemných relací. Obsahuje 14 tabulek, z nichž 4 jsou tabulky spojovací a 4 tabulky jsou číselníky. Na obrázku 4.5 je zobrazen diagram datového modelu.

Tabulka *Account* uchovává informace o přihlašovacím účtu jako jsou přihlašovací údaje, kontrolní tokeny, datum registrace apod.

Tabulka *Organizer* uchovává informace o uživateli s rolí organizátor. Jedná se o informace jako jsou souřadnice, rádius pro vyhledávání apod. Tato tabulka je spojena s tabulkou *Account*.

Tabulka *Fan* uchovává informace o uživateli s rolí fanoušek. Podobně jako organizátor obsahuje informace o souřadnicích, rádiusu, maximální cena za událost apod. Tabulka spojena s tabulkou *Account*.

Tabulka *Recommended_event* uchovává informace o uživateli a událostech, které uživatel buď vyřadil z doporučovaných přímo nebo je označil, že se mu líbí. Dále také příznak, jestli má uživatel zájem o zasílání připomínky e-mailem až se bude událost blížit. Tabulka je spojena s tabulkami *Account* a *Event*.

Tabulka *Event* uchovává veškeré informace o událostech, jako jsou název události, popis, cena, interpreti, žánry apod.

Tabulka *Coords* uchovává informace o souřadnicích a je využívána tabulkami *Fan*, *Organizer*, *City* a *Event*.

Číselník *City* obsahuje informace o městech včetně regionu, kraje a souřadnic. Je spojen s tabulkou *Event*.

Číselník Interpret uchovává informace o interpretech, jako jsou název, žánr, kontakt, webová stránka apod. Je spojen s číselníkem *Genre*.

Číselník *Genre* v sobě uchovává názvy žánrů.

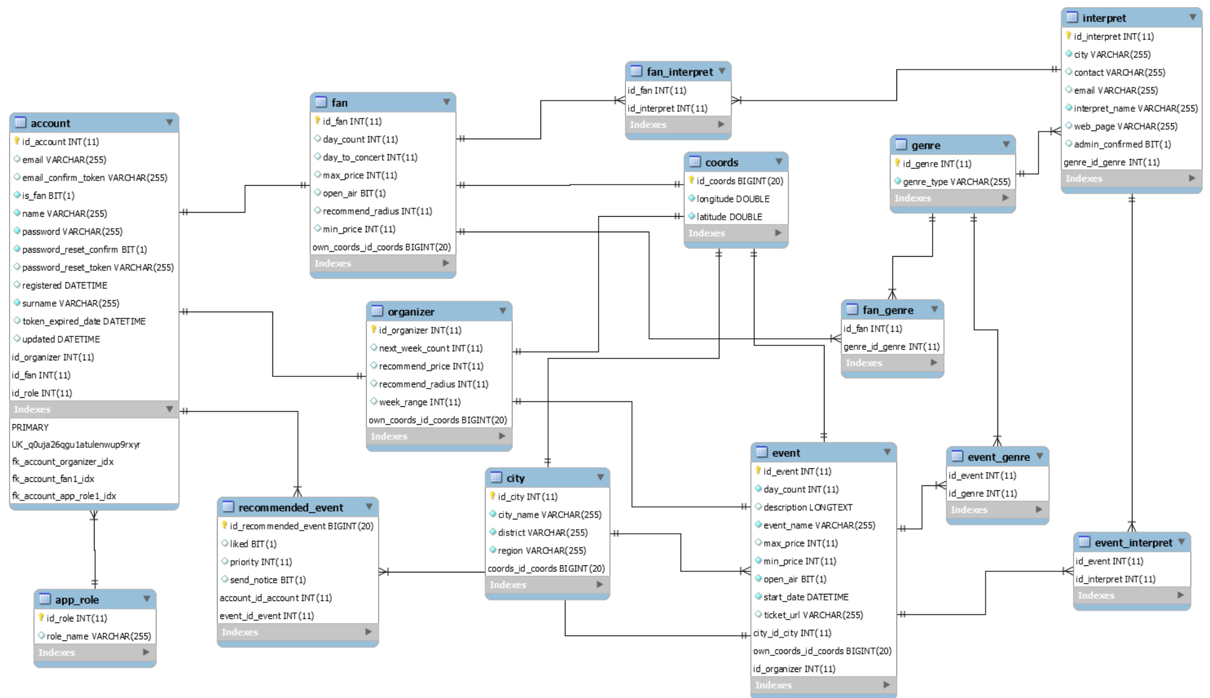
Číselník *App_*genre obsahuje názvy rolí, které jsou v aplikaci použity. Je spojen s tabulkou *Account*.

Spojovací tabulka *Fan_interpret* spojuje tabulky *Fan* a *Interpret*. Tabulka *Fan* si pomocí této tabulky uchovává informace o oblíbených interpretech.

Spojovací tabulka *Fan_genre* spojuje tabulky *Fan* a *Genre*. Tabulka *Fan* si pomocí této tabulky uchovává informace o oblíbených žánrech.

Spojovací tabulka *Event_interpret* spojuje tabulky Event a Interpret. Pomocí této tabulky si tabulka Event uchovává informace o interpretech, kteří budou na události vystupovat.

Spojovací tabulka *Event_genre* spojuje tabulky *Event* a *Genre*. Pomocí této tabulky si tabulka Event uchovává informace o žánrech.



Obrázek 4.5: Datový model aplikace. Zdroj: autor

5 IMPLEMENTACE WEBOVÉ APLIKACE

Implementace webové aplikace je založena na modelu MVC, který je popsán v kapitole 4.1 MVC. Jednotlivé části modelu jsou rozděleny na aplikační backend a frontend. Tyto dvě části mezi sebou komunikují pomocí REST rozhraní (zpracovává kontroler) a zprávy posílají výhradně ve formátu JSON.

5.1 Backend

Backend, tedy převážná logika webové aplikace, je napsán ve zjednodušené verzi Spring frameworku – Spring Boot (verze 2.1.4 RELEASE). Jedná se o RESTful API, které zpracovává pomocí kontroleru HTTP požadavky. Zahrnuje implementaci modulu *Spring starter Security*, *Spring starter data JPA* (využívající ORM nástroj Hibernate), *Spring starter mail* a *Spring starter Web* společně s *Spring starter Rest*. Dále využívá knihovny na zpracování JSONu, oAuth2 autorizaci s technologií JWT, Lombok atd. Pro uchovávání dat je zvolena MySQL databáze.

5.1.1 RESTful API

RESTful API je samostatná aplikace, která nemá žádné grafické uživatelské rozhraní. Pouze zpracovává HTTP požadavky, jako jsou GET, POST, PUT a DELETE. Je založena na architektuře REST. Tato aplikace je, jak již bylo výše zmíněno, napsána pomocí frameworku Spring Boot.

Kontroler

Aplikace má několik tříd s anotací *@RestController*. Tyto třídy jsou kontrolery (vizte architektura MVC), které zachytávají HTTP požadavky volané z klienta. Tyto požadavky jsou zpracovány a poslány do další vrstvy ke zpracování. Dle vrácených výsledků poté vrátí klientovi odpověď s výsledkem operace. Tyto třídy jsou pro přehlednost rozděleny dle typu akcí, se kterými pracují. Např třída *AuthController* zpracovává požadavky, které se týkají přihlašování, registrace a obnovy hesla. Třída *FanController* zase zpracovává požadavky, které se týkají role „fanoušek“.

Service

Kontroler po zpracování požadavku předá potřebná data další vrstvě. Tuto vrstvu představují třídy s anotací @Service. I tyto třídy jsou kvůli přehlednosti kódu rozděleny dle typů akcí podobně jako kontrolery.

5.1.2 SpringBoot Security

Jednou z důležitých částí aplikace na backendu je její zabezpečení. Není žádoucí, mohl žádat po službě data, která jsou přístupná pouze přihlášeným uživatelům. O toto se stará knihovna SpringBoot security. Ta zajišťuje zabezpečenou komunikace mezi serverem (backend) a klientem (frontend). Zde je implementována oAuth2, což je nejpoužívanější autorizační framework podporovaný Springem, společně s technologií JWT (JSON Web Token).

Knihovna *spring-boot-starter-security*, importována pomocí Maven, jako všechny SpringBoot knihovny zakončené klíčovým slovem *starter* mají základní konfigurační nastavení. V některých případech to stačí a vývojář nemusí provádět další nastavení.

WebSecurityConfig

Pro zabezpečení REST služeb je základní nastavení nedostačující, a proto je nutné implementovat vlastní konfigurační třídu *WebSecurityConfig*, která musí dědit z třídy *WebSecurityConfigurerAdapter*. Zde je nutné přetížit metodu *configure* s parametrem *HttpSecurity* jak je vidět na ukázce kódu 5.1. Metoda nastavuje následující:

- cors politiku – bezpečnostní politika umožňující sdílení zdrojů,
- csrf – bude popsáno níže,
- povolené URL včetně typu HTTP požadavku, které nevyžadují autentifikaci,
- autentifikaci pro ostatní HTTP požadavky,
- URL adresu kam přepošle http požadavek v případě chyby (např. neautorizovaný přístup),
- autentifikační filtr,
- autorizační filtr.

```
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().and().csrf().ignoringAntMatchers("/**","/login").and().authorizeRequests()
                .antMatchers(HttpMethod.POST, SIGN_UP_URL, SIGN_UP_CONFIRM_URL, GENERATE_TOKEN, RESET_PASSWORD,
NEW_PASSWORD).permitAll()
                .antMatchers(HttpMethod.GET, TEST).permitAll()
                .anyRequest().authenticated()
                .and()
                .addFilter(new JWTAuthenticationFilter(authenticationManager()))
                .addFilter(new JWTAuthorizationFilter(authenticationManager()))
                // this disables session creation on Spring Security
                .sessionManagement().enableSessionUrlRewriting(true);
    }
```

Ukázka kódu 5.1: Přetížení metody configure

CSRF

CSRF je typ útoku do internetových aplikací, který využívá důvěryhodnosti, kterou má web pro své uživatele. Pokud útočník zná URL adresy pro spuštění určitých akcí a zároveň i parametry, které se s požadavkem posílají, může využít přihlášení uživatele a podstrčit mu

vlastní připravenou stránku, která ovšem neposílá přesně to, co uživatel zadal. Uživatel provede akci, kde odešle požadavek s podstrčenými parametry a zároveň, protože byl přihlášený, tak projde autorizace a server akci provede. Právě před tímto povolení CSRF chrání. Jednou z nejjednodušších metod, která částečně chrání před CSRF útoky je například povolení pouze POST HTTP požadavků. Ty totiž nelze vyvolat pouhým aktualizováním webové stránky se zadanými URL parametry.

Zdroj [13]

UserDetailServiceImpl

Další důležitou částí, která je nutná pro zabezpečení je vlastní implementace třídy *UserDetailService*, která je označena anotací @*Service*, což pro SpringBoot znamená, že se mimo jiné jedná o tzv. „Beanu“ (vizte kapitola 4.1.3). Ta je vložena do třídy *WebSecurityConfig* pomocí anotace @*Autowired*, která zajistí její správné vložení pomocí IoC kontejneru. Tato třída musí dědit z třídy *UserDetailService* a přetížit metodu *loadUserByUsername* jak je vidět na ukázce kódu 5.2. Jako *username* slouží e-mail přihlášeného uživatele. Pokud je nalezen v databázi, vytvoří se autorizace dle role přihlašujícího se uživatele. Nakonec je vrácen vytvořený uživatel s uživatelským jménem, heslem a autorizací.

```
@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    Account applicationUser = accountRepository.findByEmail(username);
    if (applicationUser == null) {
        throw new UsernameNotFoundException(username);
    }
    GrantedAuthority authority = new SimpleGrantedAuthority(applicationUser.getRole().getRoleName());
    List<GrantedAuthority> authorities = new ArrayList<>();
    authorities.add(authority);
        return new User(applicationUser.getEmail(), applicationUser.getPassword(),true, true,
                true,true, authorities);
}
```

Ukázka kódu 5.2: Přetížení metody loadUserByUsername

Filtry

Jak bylo výše zmíněno, jsou nastaveny autorizační a autentifikační filtry.

Autorizační filtr zpracovává každý požadavek, který je podmíněn autorizací. Pokud HTTP požadavek obsahuje hlavičku *Authorization* ve správném tvaru, vpustí požadavek do dalšího zpracování. V opačném případě proces zastaví a dojde k vyvolání výjimky.

Autentifikační filtr se provede při přihlašování, kde si z HTTP požadavku získá přihlašovací jméno a heslo, které vrátí k dalšímu zpracování. Pokud bude autentifikace úspěšná, vytvoří se token, kde bude nastaveno přihlašovací jméno, expirace tokenu, a to celé zakódováno pomocí autentizačního kódu HMAC, využívajícího šifrovací algoritmus SHA-512. Tento token je vrácen klientovi po úspěšném přihlášení a je uložen do cookies. Každý HTTP požadavek, který

vyžaduje autorizace musí v HTTP hlavičce *Authorization* obsahovat tento token, kterým se ovařuje oprávnění uživatele, aby mohl provádět danou akci.

5.1.3 SpringBoot JPA

Backendová aplikace používá knihovnu *spring-boot-starter-data-jpa*. Tato knihovna vyžaduje základní nastavení v konfiguračním souboru *application.properties*, které je vidět v ukázce kódu 3.1. Bez tohoto nastavení dojde k chybě a aplikace se nespustí.

Spring data JPA poskytuje podporu úložiště pro JPA (Java Persistant API), což umožňuje vývoj aplikací, které pracují s databází. Jednou z implementací JPA je framework Hibernate.

Hibernate

Hibernate je framework napsaný v jazyce Java, který umožňuje Objektově-relační mapování (ORM). To v podstatě umožňuje objektům udržovat svůj stav i po ukončení aplikace. ORM nástroj, jak vypovídá jeho název, se stará o mapování mezi objekty a tabulkami, což znamená, že stav instance nějakého objektu vytvořeného v aplikaci je namapován na jednu entitu a uložen do relační databáze. Tento proces je obousměrný, a tedy funguje i obráceně, když je nutné získat data z databáze, tak jsou namapována a následně je vytvořen objekt, se kterým aplikace dále pracuje.

Entity

Aby JPA poznal, že má být objekt mapovaný do databáze, musí být třída označena anotací *@Entity*“ Pokud je třída označena touto anotací, pak musí splňovat následující podmínky. Třída musí obsahovat atribut s anotací *@Id*, která značí, že daný atribut je v relační databázi primárním klíčem. Další podmínkou je, aby měla třída definovaný prázdný konstruktor. (Ten je implicitně definovaný pro každý objekt, ale pokud se definuje nový, je potlačen a je nutné vytvořit i prázdný konstruktor). Poslední podmínkou takto označené třídy jsou definice metod zvaných *getter* a *setter* pro každý atribut objektu. Názvy jsou odvozené z anglických slov „get“ (získat) a „set“ (nastavit). V aplikaci ovšem mají další důležité funkce, a to převážně mapovací. Jedná se například o mapování právě mezi relační databází a samotným objektem, které tyto metody využívá, ale také o mapování z/do JSON formátu a mnoho dalšího. Pro přehlednost kódu je využita knihovna *Lombok*, která až při kompilaci automaticky vytvoří základní metody objektu, tedy gettery, settery a equals metodu (metoda sloužící pro porovnávání dvou objektů). Jediné, co je potřeba udělat je přidat třídě anotaci *@Data*. Na ukázce kódu 5.3 je zobrazeno vytvoření objektu (databázové entity) Genre, která uchovává informace o žánru.

```
@Entity
@Data
public class Genre {

  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE, generator=”sequence_genre” )
  private Integer idGenre;

  @NotNull
  @Column(unique = true)
  private String genreType;
}
```

Ukázka kódu 5.3: Definice třídy Genre

Pomocí tříd označených anotací *@Entity* je pak vytvořena relační databáze na databázovém serveru. V případě ukázky kódu 5.3 se vytvoří v databázi tabulka s názvem *Genre*, která má 2 sloupce: *idGenre*, který je primárním klíčem a jeho hodnota je vyplněna dle sekvence. Druhým sloupcem je *genreType*, který je nastaven jako unikátní a zároveň jeho hodnota nesmí být nulová (nevyplněna). O to, jak bude výsledná databáze vypadat, se postará Hibernate. Atributy objektů, které budou namapovány do relační databáze mohou obsahovat různé anotace a nejen ty, které jsou vidět na ukázce výše. Jejich nastavení a vlastnosti jsou popsány v dokumentaci Hibernate z [14]. Níže je zobrazena tabulka 5.1 základních anotací, které lze využít při vytváření entit.

Tabulka 5.1: Přehled základních anotací pro entity

Anotace	**Význam**
Id	Definice primárního klíče
GeneratedValue	Definice strategie generování id
Column	Definice sloupce (název, unikátnost, povinnost, délka apod)
NotNull	Atribut nesmí být prázdný
Max	Nastavení maximální délky pro hodnotu atributu
Email	Atribut musí splňovat formát pro Email
Lob	Jedná se o dlouhý text
Transient	Tento prvek není namapován do tabulky v databázi
ManyToOne	Relační anotace – značí relaci n:1
OneToMany	Relační anotace – značí relaci 1:n
Many to many	Relační anotace – značí relaci m:n
JoinTable	Použití s relačními anotacemi (definuje propojení tabulek)

Repozitáře

Repozitáře neboli rozhraní označené anotací @*Repository*, které dědí z rozhraní *CrudRepository* (popř. jeho variantu – v aplikaci je implementace *JPARepository*), poskytují základní CRUD funkcionality pro třídy entit, jež spravují. Význam CRUD je následující:

- C – create (vytváření)
- R – read (čtení)
- U – update (aktualizace)
- D – delete (mazání)

Pomocí repositářů lze v podstatě volat dotazy v databázi. Pro zjednodušení a přehlednost kódu JPA pracuje s tzv. JPQL (Java Persistence Query Language), což je dotazovací jazyk využívaný v Javě. Pro některé běžně využívané dotazy nad entitou lze využít předdefinovaných klíčových slov, které budou součástí názvu metody. Seznam klíčových slov lze nalézt v dokumentaci JPA[1]. Metoda musí začínat klíčovým slovem *findBy*. Poté následuje název atributu, podle kterého se hledá. Ten musí být shodný s názvem getteru (tedy pokud název getteru je *getName()*, metoda musí mít název: *findByName* a parametrem metody bude atribut *name*. Tedy celé znění metody bude:

public User findByName(String name);

V případě špatného názvu metody, ze kterého nedokáže JPQL zkonstruovat dotaz, musí být metoda označena anotací @*Query*. Pokud není anotace k dispozici, dojde k chybě. Anotace @*Query* musí obsahovat přesně definovaný dotaz do databáze. Dotaz může být dvojího typu:

- JPQL
- Native

V případě JPQL se dotaz konstruuje dle jazyka JPQL, který je poté převeden do syntaxe odpovídající konkrétní použité databázi. JPQL musí dodržovat názvy jako v Javě (název objektu velkým písmenem apod). Pokud se použije v anotaci parametr *nativeQuery=true*, tak se dotaz konstruuje přímo v syntaxi, která odpovídá použité databázi a musí být přiřazen jako textový řetězec parametru *value*. V ukázce kódu 5.4 je zobrazen způsob vyhledání uživatele dle jeho uživatelského jména.

[1] Dostupné z [*15*] v tabulce „Table 3. Supported keywords inside method names“

```
1.  Využití klíčových slov definovaných JPA
    Account findByUserName(String userName)
2.  Využití syntaxe JPQL
    @Query(„Select a from Account a where a.userName = :email“)
    Account findAccountByUserName(@Param(„userName“) String userName)
3.  Využití native query
    @Query(value=„Select * from account a where a.user_name = :userName“, nativeQuery=true)
    Account findAccountByUserName(@Param(„userName“) String userName)
```

Ukázka kódu 5.4: Porovnání metod pro dotazování do databáze

5.1.4 MySQL

V rámci aplikace je použita databáze MySQL. Při vývoji jednoduché aplikace ve SpringBootu výběr databáze nehraje příliš roli. Ve spoustě případů jsou si dotazy pro jednotlivé databáze (Oracle, MSSQL, MySQL apod) velmi podobné. Pokud se použije pro dotazy na databázi použije syntaxe JPQL, tak si s tím překladač poradí a sestrojí dotaz přímo do databáze, která je nastavena v konfiguračním souboru *application.properties*. V případě použití složitějších dotazů, či přímo procedur (procedura slouží k programování na straně DB serveru), funkcí či triggerů (trigger se vykonává, je-li nastaven, pokud nastane konkrétní akce, např. *insert*), tak už zde výběr databáze hraje velkou roli, protože syntaxe zápisu se mnohdy velmi liší. Pro vyhledávání vzdálenosti byl použit následující Haversinův vzorec[2]:

$$D = R * \cos^{-1}(\cos\varphi_1 * \cos\varphi_2 * \cos(\lambda_2 - \lambda_1) + \sin\varphi_1 * \sin\varphi_2),$$

kde:

D = celková vzdálenost,

R = poloměr země v kilometrech (pro míle je hodnota 3959),

λ1, λ2 = zeměpisná šířka 1. a 2. bodu v radiánech,

φ1, φ1= zeměpisná délka 1.a 2. bodu v radiánech

[2] Přetvořený na vzorec z [16]

V ukázce kódu 5.5 je zobrazena dotaz pro vytvoření funkce *distance* v MySQL databázi, která se používá pro výpočet vzdálenosti mezi dvěma zeměpisnými body. Výpočet vzdálenosti ve funkci *distance* odpovídá výše uvedenému Haversinovu vzorci.

```
CREATE FUNCTION `distance`(latitude double, longitude double, ownLatitude double, ownLongitude double) RETURNS double
BEGIN
  DECLARE distance double;
  set distance = 6371 * acos (
    cos ( radians(ownLatitude) ) * cos( radians( latitude ) ) *
cos( radians( longitude ) - radians(ownLongitude) ) + sin ( radIans(ownLatitude) ) * sin( radians( latitude ) ) ) ;
 RETURN (distance);
 END
```

Ukázka kódu 5.5: Definice funkce distance pro vypočítání vzdálenost mezi dvěma body

5.1.5 Doporučování událostí

V rámci celé aplikace jsou zde implementovány 2 druhy doporučování. Jeden z nich je doporučení pro organizátory událostí, druhý je doporučení pro fanoušky. Doporučování je založeno na tom, co si organizátor nebo fanoušek vyplní ve svém profilu. V tabulkách 5.2 a 5.3 jsou zobrazeny filtry, které si mohou uživatelé vyplnit pro lepší doporučování.

Tabulka 5.2: Filtry pro organizátora

Organizátor	
Filtr	**Význam**
Souřadnice	Souřadnice události, od kterých se bude počítat vzdálenost
Rádius	Vzdálenost od souřadnic, do které bude hledat fanoušky
Počet týdnů pro doporučení	Počet týdnů, z něhož se vypočítá datum, pro které bude systém doporučovat vytvoření události
Rozsah týdnů kolem data pro doporučení	Rozsah týdnů kolem data doporučené události. Pro organizátora to znamená rozsah, ve kterém bude hledat události podobného žánru nebo s podobnými interprety (při shodě pro ni sníží prioritu)
Doporučená maximální cena	Doporučená cena, kterou jsou potenciální fanoušci ochotni zaplatit za událost

Z filtrů pro organizátory z tabulky 5.2 jsou striktní rádius a doporučená maximální cena. Pokud nesouhlasí podmínky pro tyto dva filtry, pak takový fanoušek není součástí seznamu fanoušků, ze kterých se dle žánru či interpreta vytvoří doporučené události pro organizátora. V záložce doporučení pro organizátora jsou dostupné dodatečné filtry, které je možné měnit dle potřeby, aniž by se upravil profil organizátora.

Tabulka 5.3: Filtry pro fanouška

Fanoušek	
Filtr	**Význam**
Souřadnice	Souřadnice, od kterých se bude počítat vzdálenost
Rádius	Vzdálenost od souřadnic, do které bude hledat události
Dny do události	Počet dní do události
Maximální cena	Maximální cena události
Počet dní	Trvání události ve dnech
Open Air	Příznak, jestli se jedná o událost pod širým nebem
Žánry	Oblíbené žánry
Interpreti	Oblíbení interpreti

Filtry pro fanoušky z tabulky 5.3 nejsou striktní, což znamená, že při neshodě nebude událost vyřazena ze seznamu. Filtry slouží pouze pro výpočet výsledné priority mezi doporučovanými událostmi. Pokud dojde ke shodě s filtrem, zvýší se priorita. Jediný striktní filtr je vzdálenost. Pokud je vzdálenost konané události větší než rádius, pak nebude událost v seznamu doporučených událostí.

Doporučení pro organizátory

První druhem doporučování je doporučení, které je založeno na počtu fanoušků v okolí bodu, které si organizátor určí při vyplňování profilu. Na ukázce kódu 5.6 je zobrazen zdrojový kód metody pro získání doporučených událostí dle žánru. Zde se načte seznam fanoušků pomocí funkce *getFanByDistanceAndPrice*, která do databáze pošle dotaz pro vyhledání fanoušků, kteří jsou ve vzdálenosti od GPS souřadnic nastavených v profilu organizátora a mají nastavenou maximální cenu, kterou jsou ochotni dát za koncert, menší, než má nastavenou organizátor. Z tohoto seznamu se pomocí metody *getSortedMapByGenre* získá *HashMapa*, která uchovává názvy žánrů a počet fanoušků, kterým se žánr líbí. Ta je seřazena dle počtu fanoušků u daného žánru. Nakonec se dle data pro konání události, které je vypočítané z filtru, najdou všechny události, které jsou v nastaveném rozsahu dní kolem data návrhu na vytvoření události a vloží do seznamu. Pokud je seznam prázdný, dojde k jednoduchému návrhu nových událostí a ty jsou zobrazeny organizátorovi. Pokud není, tak se vytvoří návrhy pro vytvoření nové události, které jsou seřazeny dle priority a opět se zobrazí organizátorovi. Priorita se získá

tím, že se projde seznam již vytvořených událostí v okolí, ze kterého je vypočítána (dle výchozího stavu) následovně:

- Pokud již vytvořená událost nedisponuje stejným žánrem, jaký bude mít doporučená událost, je priorita zvýšena o 1.
- Vypočítá se rozdíl dní mezi daty událostí (vytvořená – doporučená k vytvoření) a výsledek je vynásoben hodnotou 0,2

Po vypočítání priority pro každou doporučovanou událost je seznam s těmito událostmi seřazen dle priority a události jsou zobrazeny organizátorovi. Výše byl popsán postup pro doporučení událostí dle žánru. Organizátor si může vybrat doporučení dle žánru nebo interpreta. Doporučení dle interpreta funguje na stejném principu jako doporučení dle žánru s tím rozdílem, že zde hraje roli interpret.

```
public List<Event> getRecommendedEventsByGenre(Account account, RecommendFilterDTO filter) {
    Organizer organizer = account.getAccountOrganizer();
    Coords ownCoords = organizer.getOwnCoords();
    List<Fan> fanInDistanceList = fanRepository.getFanByDistanceAndPrice(ownCoords.getLatitude(), ownCoords.getLongitude(),
            organizer.getRecommendRadius(), filter.getRecommendedPrice());
    HashMap<String, Integer> sortedGenreMap = getSortedMapByGenre(fanInDistanceList);
    Date eventDate = filter.getRecommendedDate();
    List<Event> nearEvents = getNearEvents(organizer, eventDate);
    if(nearEvents.isEmpty()){
        return getNewEventsByGenre(sortedGenreMap, eventDate, account);
    }else{
        return getNewEventsByPriorityGenre(sortedGenreMap, nearEvents, eventDate, account);
    }
}
```

Ukázka kódu 5.6: Zdrojový kód metody pro doporučení událostí dle žánru

Doporučení pro fanoušky

Druhým druhem doporučení je doporučení pro fanoušky, které již odpovídá doporučení na základě obsahu, jež je popsáno ve 3. kapitole. Pro fanoušky se vyhledávají události, které jsou v jejich okolí (striktní filtr). Těmto událostem je poté přiřazena priorita podle profilu uživatele. Priorita je ve výchozím stavu přiřazena dle následujících pravidel:

- Pokud je minimální cena události menší než maximální cena u fanouška, priorita se zvýší o 0,5,
- pokud je maximální cena událost menší než maximální cena u fanouška, priorita se zvýší o 0,5,
- pokud je událost dříve, než má uživatel nastaven počet dní do události, priorita se zvýší o 0,5,
- pokud je umístění události venku a zaškrtnutý příznak „openAir“ (a naopak), priorita se zvýší o 1,

- pokud je trvání události (dny) stejné, jako má uživatel nastaveno, priorita se zvýší o 1. Pokud je trvání události delší, než má uživatel nastaveno, priorita se zvýší o polovinu předchozí hodnoty (tedy 0,5),
- pokud jsou interpreti události shodní s interprety, které má uživatel mezi oblíbenými, je priorita nastavena na 1 a s každým shodným interpretem vynásobena hodnotou 1,2 (např. pokud má 3 shodné interprety, pak výsledná přiřazená priorita je 1 × 1,2 × 1,2 × 1,2 = 1,728),
- pokud jsou žánry interpretů události shodné s žánry, které má uživatel mezi oblíbenými, je priorita nastavena na 1 a s každým shodným žánrem vynásobena hodnotou 1,2 stejně jako je tomu v případě interpretů.

Níže na ukázce kódu 5.7 je zobrazen zdrojový kód výpočtu priority dle interpretů a trvání koncertu ve dnech.

```
private Double getPriorityByInterpret(Set<Interpret> interpretsEvent, Set<Interpret> interpretsFan, Configuration configuration) {
    Double priority = 1.0;
    boolean hasSameInterpret = false;
    for (Interpret interpretFan: interpretsFan){
        for(Interpret interpretEvent: interpretsEvent){
            if(interpretEvent.getIdInterpret().equals(interpretFan.getIdInterpret())){
                priority configuration.getPriorityInterprets();
                hasSameInterpret = true;
                break;
            }
        }
    }
    return hasSameInterpret?priority:0.0;
}

private Double getPriorityOnDayCount(Integer dayCountEvent, Integer dayCountFan, Configuration configuration) {
    Integer compare = dayCountEvent.compareTo(dayCountFan);
    if(compare<0) return 0.0;
    if(compare==0) return configuration.getPriorityDayCount();
    return configuration.getPriorityDayCount() / 2.0;
}
```

Ukázka kódu 5.7: Ukázka výpočtu priority

Výše uvedené hodnoty, které slouží k výpočtu priority události pro fanouška i organizátora, jsou hodnoty výchozího stavu konfigurace priorit aplikace.

5.1.6 Plánovač

Aplikace využívá technologie *Spring Scheduling*. Jedná se o technologii, která zajistí plánování pravidelných úloh, které se v aplikaci budou vykonávat v nějakém intervalu. Je to využito k zasílání upozornění o nastávajících událostí fanouškům, kteří si je přidali do svého seznamu.

Pro zajištění plánování je nutné v konfigurační třídě (jakákoliv třída s anotací @*Configuration*) vložit anotace @*EnableScheduling*. Tato anotace umožňuje nastavit anotaci @*Scheduled* nad metodu, která se má pravidelně provádět. Anotace vyžaduje parametr *cron*, kde je nastaven předpis pro plánování. V ukázce kódu 5.8 je zobrazeno nastavení plánovače, který má spouštět metodu *sendNotice()* vždy ráno v 6 hodin.

```
@Scheduled(cron = "0 0 6 * * *")
public void sendNotice() {
      …
}
```

Ukázka kódu 5.8: Nastavení plánování

5.2 Frontend

Frontend, tedy to, co vidí a s čím interaguje uživatel aplikace, je napsaný v JavaScript frameworku React.js. Jak je zmíněno v kapitole 4.3, která popisuje React, celá stránka je složena z několika komponent, které jsou po zkompilování přetvořeny na HTML stránku. Komponenty jsou použity převážně z knihovny *reactstrap*, ze které se používají různé formulářové prvky jako jsou tlačítka, textová pole apod. Tato knihovna používá kaskádové styly (CSS) na základě knihovny *Bootstrap*, která je použita napříč webovou aplikací pro stylování stránky. Dále pro jednotlivé komponenty jako například tabulky, výběrová menu apod., které má knihovna „reactstrap“ implementované pouze v základu (např. u tabulek nemají implementaci stránkování) nebo vůbec, ale také komponenty pro práci s google mapami, byly zvoleny jednotlivé knihovny, které jsou pro tyto implementace specializovány (např. knihovna *react-select*).

Volání služeb REST API, tedy http požadavky poslané na server, jsou provedeny pomocí služby *axios* ze stejnojmenné knihovny. Ta slouží k volání služeb různých REST API. Lze s ní volat http požadavky (POST, GET apod) včetně připojení hlavičky, či parametrů. Úspěšná odpověď ze serveru je zpracována v bloku *then*. V případě, že dojde k chybě, je proces přesměrován do bloku *catch*. Axios nabízí i blok *finally*, kde může být vložena implementace, která se provede ať už odpověď bude úspěšná nebo skončí chybou. V ukázce kódu 5.9 je zobrazeno použit axios pro volání služeb REST API.

```
const data = this.getAccount();
   axios.post(SERVER_URL + REGISTER, data, {
      headers: {
         'content-type': 'application/JSON;charset=utf-8',
      }
   }).then(response => {
         this.props.history.push("/login");
         toast.success('Registrace proběhla úspěšně, {
            position: toast.POSITION.TOP_LEFT,
         })
   }).catch(error => {
      console.error(error)
      toast.error(Registrace se nezdařila.', {
         position: toast.POSITION.TOP_LEFT,
      })
   })
```

Ukázka kódu 5.9: Ukázka použití služby axios k volání REST API

Celá React aplikace je vložena do html stránky index.html, která má v elementu *<body>* element *<div>* s id *root*. Tento element je načten jako DOM objekt v souboru *index.js*. Do tohoto elementu je „vložen" element *Root*, obsahující stromovou strukturu komponent, který je pomocí ReactDOM vykreslen, jak je vidět na ukázce kódu 5.10. Na ukázce je ještě vytvořen objekt *store* a nastavený tzv. *axios interceptor* jako síťová služba. Jejich význam bude probrán níže v této kapitole.

```
const store = createStore(rootReducer, applyMiddleware(thunk));
const Root = () => {
   return (
      <Provider store={store}>
         <Router >
            <App/>
         </Router>
      </Provider>
   );
};
const history = createBrowserHistory();
NetworkService.setupInterceptors(store, history);
const rootEl = document.getElementById('root');
ReactDOM.render(<Root/>, rootEl);
```

Ukázka kódu 5.10: Ukázka vytvoření úvodní stromové struktury react aplikace

Celá aplikace je vložena do komponenty *App*, která je v podstatě kořenovou komponentou celého stromu. Ta je pouze obalena komponentami „Provider" (bude probráno později společně s významem objektu *store*) a *Router*. Komponenta Router musí obalovat celou komponentu *App*, resp. ty komponenty aplikace, které přesměrovávají URL adresy (routy = cesty). Komponenta *App* obaluje komponenty *ToastContainer*, která se stará o vypisování zpráv, které informují o úspěšných operacích nebo případných chybách, které v rámci aplikace nastanou. Dále obsahuje komponentu *Header*, která obsahuje horní menu pro pohyb na webových stránkách, a nakonec důležitou komponentu *Routes*, vizte ukázka kódu 5.11.

```
render() {
  return (
    <div className="App">
      <ToastContainer autoClose={5000} closeOnClick={true}/>
      <Header/>
      <Routes/>
    </div>
  );
}
```

Ukázka kódu 5.11: Ukázka obsahu komponenty App

5.2.1 Routes

Komponenta *Routes* se stará o směrování aplikace. Všechny cesty v URL adrese v rámci aplikace zachytí a dle těchto cest, které porovná s atributem *path*, přesměrují aplikaci a zajistí překreslení obsahu dle přiřazené komponenty:

- Route: klasická komponenta z knihovny *react-router-dom*, která zajistí překreslení obsahu dle URL adresy, parametry jsou *path* (cesta) a *component* (komponenta pro vykreslení)
- PrivateRoute: komponenta z knihovny *react-private-route*. Od komponenty *Route* se liší převážně tím, že vyžaduje parametr *isAuthenticated*. Některé cesty jsou přístupné pouze přihlášeným uživatelům a nepřihlášení by k nim neměli mít přístup. To zajistí tato komponenta, která nepřihlášeného uživatele přesměruje na přihlašovací stránku.

5.2.2 Redux

Redux je open-source JavaScript knihovna, která spravuje aplikační stav. Jedná se o centralizovaný kontejner, kde jsou uchovávány stavy aplikace, které se používají napříč komponentami v celé aplikaci. Může se například jednat o příznak *isAuth*, který značí, že je uživatel přihlášený. Aby se tento příznak nemusel předávat od komponenty ke komponentě, která jej skutečně potřebuje pro nějaké omezení, ve vlastnosti *props*, uchovává se tento příznak v tzv. *store* objektu, který je definovaný v souboru *index.js* a vložen komponentě *Provider*, která obaluje celou aplikaci. Komponenta, která pak potřebuje nějaký objekt ze *store* objektu jej pomocí objektu *mapStateToProps* a funkce *connect* připojí do *props* a k tomuto objektu přistupuje pomocí tečkové notace: *this.props.isAuth* podobně jako by byl tento příznak vložen komponentě jejím rodičem jako parametr, což je vidět na ukázce kódu 5.12.

```
<AccountMenu isAuth={this.props.isAuth}/>

const mapStateToProps = state => {
   return {
      isAuth: state.auth.isAuth,
   }
};
export default connect(mapStateToProps)(Header);
```

Ukázka kódu 5.12: Zobrazení připojení příznaku isAuth do props ze store včetně použití

Výhodou použití Reduxu je, že jsou všechny potřebné stavy v aplikace na jednom místě a lze k nim velmi jednoduše přistoupit. Pro správné využití Reduxu je nutné definovat tzv. „action types“ a „reducers“ (dále akční typy a reduktory). Stavy ve *store* jsou neměnné (tzv. „immutable“), což znamená, že je nelze změnit přímo. Jdou měnit pouze pomocí předem definovaných metod, které vyvolávají akční typy a případně jim dle potřeby předají potřebné parametry. Takto vyvolané akční typy jsou poté zpracovány pomocí reduktoru, který změní stav v objektu *store*. Tento stav zajistí aktualizaci stavu komponenty v úplném kořenu stromové struktury aplikace, čímž se zajistí aktualizace všech komponent ve stromové struktuře, což je obrovská výhoda ve využití Reduxu. Na ukázce kódu 5.13 je zobrazeno definování akčních typů pro přihlašování.

```
export const LOGIN_START = 'LOGIN_START';
export const LOGIN_SUCCESS = 'LOGIN_SUCCESS';
export const LOGIN_FAILURE = 'LOGIN_FAILURE';
export const LOGOUT = 'LOGOUT';
export const LOADING_START = 'LOADING_START';
export const LOADING_STOP = 'LOADING_STOP';
```

Ukázka kódu 5.13: Ukázka definice akčních typů

Na ukázce kódu 5.14 je poté zobrazeno vyvolání akcí. Akce, resp. metody, které vrací akční typ reduktoru, se vyvolávají pomocí tzv. „dispatcheru“. Dispatcher pomocí metody *dispatch*, která vyžaduje jako parametr název volané metody akčního typu, vyvolá akci, kterou poté zpracuje reduktor a dojde k úpravě stavu celé aplikace.

```
dispatch(loginSuccess(data));
 export function loginSuccess(data) {
   return {
      type: LOGIN_SUCCESS,
      role: data
   }
}
```

Ukázka kódu 5.14: Ukázka definice metody, vracející akční typ reduktoru a její vyvolání

Ukázka kódu 5.15 zobrazuje definování reduktoru *AuthReducer*, který zpracovává vyvolané akční typy. Dle vyvolaného typu a případných dostupných parametrů (např. parametr role) předem definovaným způsobem upraví stav aplikace.

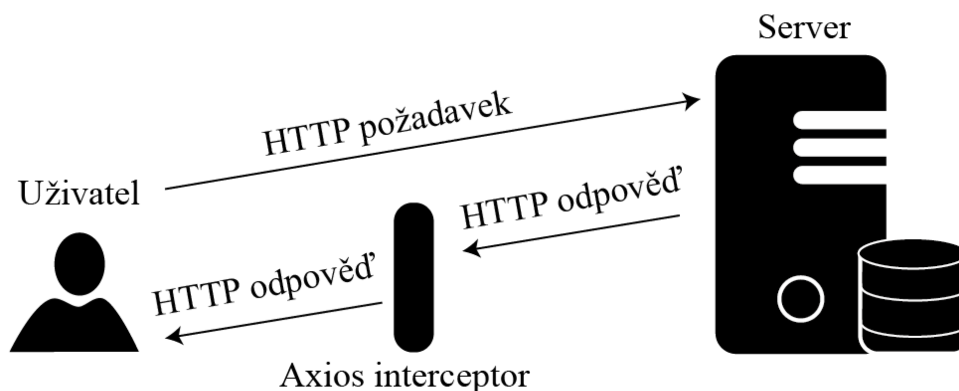
Zdroj [17]

```
export function AuthReducer(state = { loading: false, isAuth: false, role: "", resetPassword: false }, action) {
  const { type, role} = action;
  switch (type) {
    case LOGIN_START:
      return { ...state, loading: true, resetPassword: false };
    case LOGIN_SUCCESS:
      return { ...state, loading: false, isAuth: true, role: role };
    case LOGIN_FAILURE:
      return { ...state, loading: false, resetPassword: true };
    case LOGOUT:
      return { ...state, isAuth: false, role: null};
    case LOADING_START:
      return { ...state, loading: true};
```

Ukázka kódu 5.15: Ukázka definice reduktoru

5.2.3 Axios interceptor

Jak bylo zmíněno na začátku kapitoly, v rámci volání služeb REST API je použit *Axios interceptor*. Zjednodušeně se jedná o mezičlánek mezi odpovědí serveru na požadavek klienta a zpracováním odpovědi klientem. Graf na obrázku 5.1 zobrazuje jeho funkcionalitu.



Obrázek 5.1: Ukázka funkcionality axios interceptoru. Zdroj: autor

Klient (uživatel) prostřednictvím HTTP volání služby REST API pošle požadavek. Server jej zpracuje a pošle zpět odpověď klientovi. Tato odpověď je nejdříve zpracována interceptorem. Ten může na základě HTTP stavů vyvolat nějakou akci, upravit stav apod. nakonec pokud je to žádoucí, pošle odpověď dál klientovi, resp. komponentě, která vyvolala požadavek a čeká na tuto odpověď. V aplikaci je tato funkcionalita použita k zachycení HTTP stavu s kódem 401, což je tzv. „Forbidden“ a značí neoprávněný přístup. To může být například způsobeno dlouhodobou neaktivitou uživatele, kterému vypršel token. Když pak zavolá službu, která vyžaduje autorizaci, tak je vrácen právě tento stav a dojde k odhlášení z aplikace a přesměrování na přihlašovací stránku.

5.2.4 Přihlašování a registrace

Pro využití služeb portálu potřebuje uživatel mít zřízený uživatelský účet. Pokud je již registrován, může se přihlásit. V opačném případně se musí registrovat do portálu. Může se registrovat buď jako organizátor, který bude zakládat události v okolí, nebo jako fanoušek, který naopak tyto události bude vyhledávat. Níže na obrázku 5.2 je zobrazen registrační formulář včetně validace, společně s přihlašovacím formulářem.

Obrázek 5.2: Registrační a přihlašovací formulář. Zdroj: autor

5.2.5 Profil uživatele

V přílohách A a B je zobrazen formulář pro vyplnění profilu fanouška a organizátora. Takto vyplněný profil je poté použit pro přesnější doporučování událostí. Uživatel si může dle své libosti vyplnit profil včetně označení bodu v mapě, kterou si může libovolně přibližovat a oddalovat. Automaticky se mu generují souřadnice při změně bodu, které nemůže změnit sám, ale pouze pomocí přesunutí bodu.

5.2.6 Vytvoření události

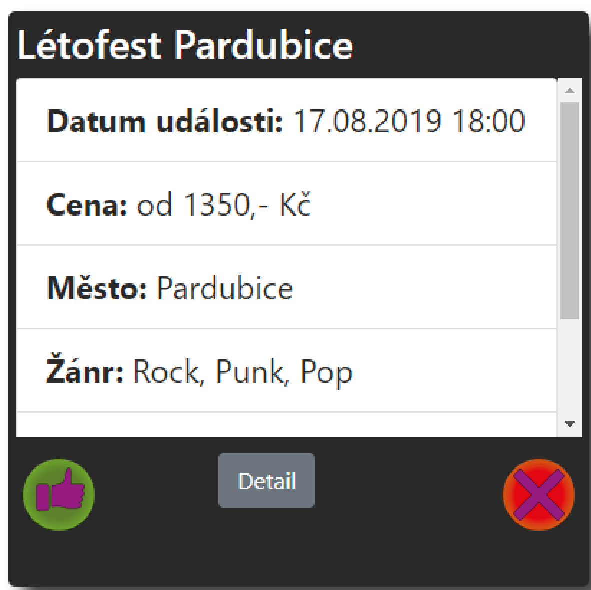
Organizátor může událost vytvořit dvojím způsobem:

1. V kartě „Moje události“ má zobrazené události, které již založil včetně přepínače, který zobrazí i události z historie. Zde může kliknout na tlačítko „Vytvořit událost“, čímž se mu otevře dialogové okno, kde vyplní informace o konané události a uložit ji.
2. V kartě „Doporučené události“ si může nechat vygenerovat doporučené události, kde mu aplikace napoví, kolik fanoušků v okolí jeho zvoleného bodu poslouchá daný žánr, či interpreta a podle toho se může rozhodnout. Rozkliknutí karty s nabízenou události se mu opět otevře dialogové okno s možností vyplnit informace o konané události a uložit jí.

Takto vytvořené události ještě může upravovat v případě, že na nich nejsou ještě přihlášení žádní uživatelé (nedali této události, že se jim líbí). Poté může upravit pouze některé informace.

5.2.7 Registrace k události

Fanoušek může procházet události, které mu aplikace doporučí dle jeho vyplněného profilu. V kartě „Doporučené události“ se zavolá služba, která vrátí doporučené události pro uživatele. Uživatel pak vidí karty, kde jsou základní informace o události. Může si zobrazit její detail, kde jsou zobrazeny všechny dostupné informace o události. Dále může události vyřadit z doporučování stisknutím tlačítka s křížkem případně vložit mezi oblíbené události pomocí tlačítka s palcem. Na obrázku 5.3 je zobrazena karta s doporučenou událostí.



Obrázek 5.3: Ukázka karty doporučené události. Zdroj: autor

5.2.8 Vložení interpreta

Aplikace umožňuje uživatelům vkládat interpreta, který v databázi chybí. Pokud chce organizátor vytvořit událost a chybí mu tam interpret, který na události bude vystupovat, či pokud si chce fanoušek přidat nového interpreta do svého seznamu, může v sekci interpreti (menu události) interpreta vytvořit. Systém mu umožní vyplnit formulář a interpreta uložit. Pokud má uživatel roli administrátora, tak se interpret rovnou uloží a poté se zobrazuje v seznamech při výběru interpretů. Pokud ne, tak se interpret založí a čeká na potvrzení od administrátora. Tomu se odešle email s informací o tom, že byl uložen požadavek na vložení nového interpreta. Ten může následně zkontrolovat vyplněné informace a nového interpreta potvrdit, čímž se začne zobrazovat v seznamech podobně jako by jej rovnou vložil on sám.

5.2.9 Administrace

Jak již bylo výše zmíněno, aplikace umožňuje i roli administrátora. Ten mimo potvrzování požadavků na vložení nového interpreta a vkládání nového interpreta sám, může zobrazit i seznam žánrů a zároveň vkládat nové žánry, prohlížet seznam uživatelů a jednotlivých událostí, aby měl přehled o počtu uživatelů, kteří systém využívají a zároveň měl přehled o jednotlivých událostech. Tyto seznamy má zobrazené v tabulce, která je z důvodu velkého množství dat filtrována a stránkována na straně serveru. Dále má k dispozici službu, kterou při spuštění aplikace do provozu může nahrát seznam měst včetně jejich souřadnic, aby uživatelé mohli lépe zaměřovat souřadnice. Města jsou uložená v souboru *cities.csv*[3].V případě nutnosti může města přidat.

Administrátor dále může zobrazit konfiguraci aplikace, ve které může upravit hodnoty, které jsou použity při výpočtu priority pro jednotlivé filtry[4].

[3] Jedná se o upravený soubor dostupný z [18]

[4] Při implementaci byla v databázi vytvořena nová tabulka „*Configuration*“, kde jsou tyto hodnoty uloženy. Tabulka se ukázala být důležitá v případě potřeby změnit hodnoty pro výpočet priority aniž by bylo potřeba překompilovat aplikaci a restartovat ji. Z tohoto důvodu není zobrazena v datovém modelu v kapitole 4. Analýza webové aplikace.

ZÁVĚR

Úkolem mé diplomové práce bylo popsat doporučovací systém a poté navrhnout a implementovat aplikaci, která bude sloužit jako doporučovací systém pro hudební události v blízkém okolí. Aplikace měla být typu klient-server, kdy server bude jako REST API napsané ve SpringBootu a klient webová aplikace napsaná v React.js.

V teoretické části jsou popsány hudební aplikace včetně alternativ k té, kterou jsem vyvíjel a zároveň je popsán doporučovací systém s detailním zaměřením na doporučování na základě obsahu. Následující kapitola popisuje technologie, které jsem v rámci aplikace využíval. Dále je popsána analýza a návrh aplikace, kde jsou rozepsány funkční a nefunkční požadavky, případy užití a jsou nastíněny vztahy mezi třídami pomocí analytických tříd, a nakonec detailní popis datového modelu včetně grafu, který zobrazuje tabulky a relace mezi nimi.

V poslední kapitole popisuji vyvíjenou webovou aplikaci, resp. aplikace, které jsou rozděleny na server a klienta. Při vývoji jsem nejdřív implementoval zabezpečení v REST API a poté v Reactu připravil vše potřebné k tomu, abych mohl volat služby, využívající autorizaci a autentizaci. Byl tedy implementován Redux, díky kterému si aplikace drží potřebné stavy, společně s registrací uživatele a přihlašováním. Poté jsem postupně přidával funkce pro roli organizátora a následně pro roli fanouška. Souběžně s tím byly implementovány i funkce na backendu včetně nahrání testovacích dat a doporučování. Nakonec byly implementovány funkce pro administrátora a celá funkčnost aplikace otestována.

Vývoj aplikací mi přinesl mnoho praktických poznatků o komunikaci mezi klientem a serverem. Dále poznatky o rizicích a výjimečných stavech ve vývoji takovýchto aplikací jako je zabezpečení, možné útoky, neobvyklé situace apod. a v neposlední řadě jsem se díky této práci naučil s JavaScript frameworkem React.js, se kterým jsem do té doby neměl téměř žádné zkušenosti.

Aplikace je mimo implementace doporučovacího systému rozšířena o několik dalších základních funkcionalit, které vyžaduje vývoj webových aplikací (např. zabezpečení, validace formulářů apod.). Pro budoucí využití aplikace je ještě třeba doplnit další funkce tak, aby se uživatelům s aplikací lépe pracovalo. Jedná se například o lepší využití zabezpečení, které SpringBoot nabízí, vizualizace či zachycení více výjimek, které mohou nastat. Dále rozšíření administrace o nové funkce jako je vytvoření nového administrátora, zobrazení statistik, rozšíření konfigurace administrátorem, apod.

POUŽITÁ LITERATURA

[1] JANNACH, Dietmar. *Recommender systems: an introduction*. New York: Cambridge University Press, 2011. ISBN 978-0-521-49336-9.

[2] AGGARWAL, Charu C. *Recommender systems: the textbook*. New York, NY: Springer Science Business Media, 2016. ISBN 978-3319296579.

[3] *Tutorialspoint: simply easy learning* [online]. 4th Floor, Incor9 Building, Kavuri Hills, Madhapur, Hyderabad, Telangana 500081, 2012. Dostupné z: http://www.tutorialspoint.com/

[4] *Spring framework: Reference documentation* [online]. 2010. Dostupné z: https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/

[5] *Spring Boot Reference Guide* [online]. 2018. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

[6] *Apache Maven Project* [online]. 2003. Dostupné z: https://maven.apache.org/

[7] *Rest API tutorial: What is REST* [online]. 2019. Dostupné z: https://restfulapi.net/

[8] THOMAS FIELDING, Roy. *Architectural Styles and the Design of Network-based Software Architectures*. IRVINE, 2000. Disertační práce. UNIVERSITY OF CALIFORNIA.

[9] *React: A JavaScript library for building user interfaces* [online]. 2019. Dostupné z: https://reactjs.org/

[10] HINKULA, Juha. *Hands-On Full Stack Development with Spring Boot 2.0 and React*. Birmingham: Packt Publishing, 2018. ISBN 978-1-78913-808-5.

[11] ČÁPKA, David. *MVC architektura* [online]. 2013, , 1. Dostupné z: https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor

[12] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

[13] *Spring Security Reference* [online]. 2004. Dostupné z: https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/

[14] *Hibernate Annotations* [online]. 2010. Dostupné z: https://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single/

[15] *Spring Data JPA – Reference Documentation* [online]. 2008. Dostupné z: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

[16] FUCHS, Jochem. The best way to locate, in MySQL 8. Medium [online]. 2018. Dostupné z: https://medium.com/maatwebsite/the-best-way-to-locate-in-mysql-8-e47a59892443

[17] *Redux: A predictable state container for JavaScript apps* [online]. 2015. Dostupné z: https://redux.js.org/

[18] ZEMEK, Michal. souradnice-mest. *GitHub* [online]. San Francisco: GitHub, 21 Mar 2012. Dostupné z: https://github.com/33bcdd/souradnice-mest

PŘÍLOHY

Příloha A – Editace profilu 68

Příloha B – Editace profilu..69

Příloha C – Použité knihovny backend 70

Příloha D – Použité knihovny frontend..71

PŘÍLOHA A – EDITACE PROFILU

Editace profilu role Fanoušek

Nastavení profilu

Email

st46540@student.upce.cz

Jméno

Radim

Příjmení

Bednář

Změnit heslo

Zadejte polohu

Zeměpisná šířka

50.1272005

Zeměpisná délka

15.856539200000043

Radius pro doporučování (km)

20

Maximální cena události

1500

Počet dní do události

20

Trvání koncertu (dny)

1

Hudební žánr

Rock × Country × Pop ×

Interpret

Rybičky 48 × Divokej Bill × Mandrage × Kabát ×

Open air

Uložit profil

PŘÍLOHA B – EDITACE PROFILU

Editace profilu role Organizátor

Nastavení profilu

Email

radim.bednar@student.upce.cz

Jméno

Radim

Příjmení

Bednář

Změnit heslo

Zadejte polohu

Zeměpisná šířka

50.07278009999999

Zeměpisná délka

15.802525500000002

Radius pro doporučování (km)

15

Výchozí počet týdnů pro doporučení

4

Rozsah týdnů kolem data pro doporučení

2

Doporučená maximální cena

850

Uložit profil

PŘÍLOHA C – POUŽITÉ KNIHOVNY BACKEND

Seznam knihoven, které jsou importované pomocí nástroje Maven

Název knihovny	**Verze**
org.springframework.boot.spring-boot-starter-parent	2.1.4.RELEASE
org.springframework.boot.spring-boot-starter-data-jpa	2.1.4.RELEASE
org.springframework.boot.spring-boot-starter-data-rest	2.1.4.RELEAS
org.springframework.boot.spring-boot-starter-web	2.1.4.RELEASE
org.springframework.boot.spring-boot-starter-security	2.1.4.RELEAS
org.springframework.boo.spring-boot-starter-mail	2.1.4.RELEASE
org.springframework.boot.spring-boot-cache	2.1.4.RELEASE
org.springframework.boot.spring-boot-devtools	2.1.4.RELEASE
com.auth0.java-jw	3.7.0
mysql.mysql-connector-java	
io.jsonwebtoken.jjwt	0.9.0
org.json.json	20090211
com.google.code.gson.gson	2.8.5
org.projectlombok.lombok	1.18.6

PŘÍLOHA D – POUŽITÉ KNIHOVNY FRONTEND

Seznam knihoven, které jsou přidané v React.js aplikaci

Knihovna	Verze
@fortawesome/fontawesome-svg-core	^1.2.15
@fortawesome/react-fontawesome	^0.1.4
@types/googlemaps	^3.36.4
@types/markerclustererplus	2.1.29
@types/react	^16.0.0
axios	^0.18.0
bootstrap	^4.3.1
jquery	1.9.1
lodash	^4.17.11
match-sorter	^3.1.1
popper.js	1.14.7
React, react-dom	^16.8.3
react-geocode	^0.1.2
react-google-autocomplete	^1.1.0
react-google-maps	^9.4.5
react-private-route	^1.1.2
react-redux	5.0.7
react-router	^4.3.1
react-router-dom	^4.3.1
react-scripts	^2.1.8
react-select	^2.4.2
react-spinners	^0.5.4
react-table	^6.10.0
react-toastify	^4.5.2
reactstrap	^7.1.0
Reactstrap-confirm	^1.1.0
redux	4.0.0
redux-thunk	2.3.0
universal-cookie	^3.1.0